Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Tutorial de Transferência de Estilo Artístico como Ferramenta Didática de Introdução ao Deep Learning na Disciplina de Introdução ao Processamento de Imagens

Filipe Schulz dos Santos

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientador
Prof. Dr. Alexandre Zaghetto

Brasília
2018

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Computação — Licenciatura

Coordenador: Prof. Dr. Wilson Henrique Veneziano

Banca examinadora composta por:

Prof. Dr. Alexandre Zaghetto (Orientador) — CIC/UnB
Prof. Dr. Bruno L. Macchiavello Espinoza — CIC/UnB
Prof. Dr. Marcus Vinícius Chaffim Costa — FGA/UnB

CIP — Catalogação Internacional na Publicação

dos Santos, Filipe Schulz.

Tutorial de Transferência de Estilo Artístico como Ferramenta Didática de Introdução ao Deep Learning na Disciplina de Introdução ao Processamento de Imagens / Filipe Schulz dos Santos. Brasília : UnB, 2018. 167 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2018.

1. ensino, 2. processamento de imagens, 3. machine learning, 4. deep learning, 5. transferência de estilo artístico

CDU 004.8

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Tutorial de Transferência de Estilo Artístico como Ferramenta Didática de Introdução ao Deep Learning na Disciplina de Introdução ao Processamento de Imagens

Filipe Schulz dos Santos

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Prof. Dr. Alexandre Zaghetto (Orientador)
CIC/UnB

Prof. Dr. Bruno L. Macchiavello Espinoza
CIC/UnB

Prof. Dr. Marcus Vinícius Chaffim Costa
FGA/UnB

Prof. Dr. Wilson Henrique Veneziano
Coordenador do Curso de Computação — Licenciatura

Brasília, 13 de Julho de 2018

Dedicatória

Àquele que trouxe a luz à existência, e com ela todas as cores, e nos fez todas as suas maravilhas. *Soli Deo Gloria.*

Agradecimentos

À minha esposa amada, carne da minha carne e osso dos meus ossos. À minha família, minha mãe, meu pai e meus irmãos, e meus amigos. Aos professores que muito me ensinaram e inspiraram, em especial ao caro orientador. A todos esses, obrigado pelo apoio, dedicação e inspiração ao longo de todos esses (muitos) anos. Ombros de gigantes, etc.

Ser, digamos, intediável, é a chave da vida moderna. Se você é imune ao tédio, não há nada, literalmente, que você não possa alcançar

- David Foster Wallace

Resumo

O crescimento sempre constante do poder de processamento computacional disponível à comunidade acadêmica tem tornado possível a utilização de modelos de *deep learning* em diversos domínios, entre eles a visão computacional e o processamento digital de imagens, e a presença crescente dessas tecnologias tanto em contextos aplicados quanto no dia-a-dia do cidadão comum tornam cada vez maior a necessidade de inclusão desse tópico no ensino acadêmico da área de processamento de imagens digitais. O objetivo do presente trabalho é propor um tutorial que apresente conceitos de Deep Learning por meio de uma aplicação expecífica, de transferir o estilo artítistico de uma imagem para o conteúdo de outra, e que possa complementar o ensino da disciplina de Introdução ao Processamento de Imagens Digitais, visando enriquecer e expandir o aprendizado teórico obtido em sala de aula.

Palavras-chave: ensino, processamento de imagens, machine learning, deep learning, transferência de estilo artístico

Abstract

The constant growth of computational processing power available to the academic community has made possible the application of deep learning models to many domains, amongst them the computational vision and the digital image processing, and the growing presence of these technologies both in specific contexts and in the day-to-day life of the average person makes more important than ever the inclusion of this topic in the curriculum of digital image processing classes. The goal of this work is to propose a tutorial that presents Deep Learning concepts through a specific application, that of transfering the artistic style of an image to the content of another, to be used within the Introduction to the Digital Image Processing discipline, seeking to complement and enrichen the classroom theoric learning.

Keywords: teaching, image processing, machine learning, deep learning, art style transfer

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Por muito tempo, o uso de computadores para estudar e manipular imagens digitais foi restrito a um seleto grupo de especialistas com acesso a equipamentos caros e complexos, combinação comumente encontrada apenas em laboratórios de grandes universidades e empresas [1]. Apesar disso, o campo da Visão Computacional tem se tornado cada vez mais importante e eficaz nos últimos anos, devido à amplitude de suas aplicações em áreas tão diversas como vigilância e monitoramento, saúde e medicina, esportes e recreação, robótica, *drones* e carros autodirigidos. Tarefas de reconhecimento visual, como classificação, localização e detecção de imagens são os fundamentos de muitas dessas aplicações. Os recentes avanços no campo específico do Aprendizado de Máquina tem levado a performances cada vez melhores na execução de tarefas e sistemas de reconhecimento visual [2]. Aprendizado de máquina não é apenas mais um jargão da indústria e da academia, mas uma ferramenta poderosa que tem aberto um vasto leque de possibilidades dentro da área de visão computacional - essa arte e a ciência de fazer máquinas entenderem padrões e representações de imagens e vídeos em alto nível, tornando-as capazes de tomar decisões inteligentes simulares às que um ser humano tomaria [3].

1.1 Problema

Se a aplicação dessas técnicas de processamento de imagens e visão computacional do estado da arte já são cada vez mais presentes no cotidiano, seu estudo formal acadêmico também é uma realidade. Em proposta curricular de 2005, a SBC já sugeria o estudo da disciplina como parte fundamental do currículo dos cursos de Ciências e Engenharia da Computação [4]. Desde o mesmo ano, o ENADE, exame nacional de avaliação dos cursos de graduação, já vem cobrando questões a respeito do assunto nas provas dos cursos de computação [5]. Em 2017, o Ministério da Educação, com apoio da SBC, homologou um documento de Diretrizes Curriculares Nacionais para os cursos de Bacharelado em Ciência da Computação, Bacharelado em Engenharia de Computação, Bacharelado em Engenharia de Software, Bacharelado em Sistemas de Informação e de Licenciatura em Computação [6]. Nele, a disciplina de Processamento de Imagens consta como conteúdo curricular de formação tecnológica básica para todos os cursos de Bacharelado e de Licenciatura abordados no documento.

Entretanto, por mais que essa seja uma disciplina visualmente intuitiva, muitos dos conceitos e técnicas que a embasam são abstratos e suas expressões são puramente ma-

temáticas. Conversão de espaços de cores, Transformadas de Fourier no domínio da frequência e equalização de histogramas são apenas alguns exemplos de tópicos abordados no decorrer de um curso inicial de processamento de imagens que podem soar obscuros e talvez sem propósito para um aluno, caso não haja exemplos claros e demonstrações práticas. Tópicos avançados como redes neurais e o chamado *deep learning* são relegados aos últimos momentos dessas disciplinas, e acabam sendo abordados de forma rápida e superficial.

Apesar disso, retornando às diretrizes do MEC, o aluno egresso do curso de Licenciatura "é precipuamente responsável por apresentar a Computação como ciência à escola, e por consequência à sociedade. Um dos seus papéis é, portanto, o de ajudar a construir narrativas capazes de dar sentido e significado aos conceitos e fenômenos da Computação, especialmente onde esses fenômenos e conceitos se apresentarem contra-intuitivos ou de difícil composição com as narrativas consensuais." [6].

1.2 Proposta

Uma vez que é esperado de um aluno formado em Licenciatura uma atuação de "ponte" entre conceitos e fenômenos abstratos e os alunos que os estudam, levantou-se a seguinte proposta: *a criação de um tutorial que aborde esses assuntos - muitas vezes negligenciados, por falta de tempo, principalmente, no ensino das disciplinas de processamento de imagens e visão computacional - de forma atrativa ao aluno, que sirva como uma ferramenta de estudo dirigido autodidático, pode complementar, enriquecer e expandir o aprendizado teórico obtido em sala de aula.*

1.3 Objetivo

Diante do contexto apresentado e dos problemas levantados, deseja-se ao longo desse trabalho: **elaborar um tutorial que apresente conceitos de Deep Learning por meio de uma aplicação de transferência de estilo artístico de uma imagem para o conteúdo de outra**, visando apresentar aos alunos um conteúdo não visto em sala de aula, mas baseado nos fundamentos nelas explanado.

1.4 Organização do Documento

No Capítulo 2 é apresentada uma revisão bibliográfica, composta da fundamentação teórica por trás do conteúdo a ser abordado no tutorial e a própria decisão de apresentar esse conteúdo em forma de um tutorial autocontido, e de uma breve análise de alguns trabalhos correlatos. O Capítulo 3 apresenta o tutorial em sua forma completa, como disponibilizado para o aluno. O Capítulo 4 apresenta os resultados experimentais da aplicação prática do tutorial em ambiente de ensino e também dos experimentos realizados com o algoritmo explicado no tutorial. O capítulo encerra o trabalho com sua conclusão, considerações finais e propostas para trabalhos futuros. Materiais e resultados de apoio foram inclusos como apêndices ao final do documento.

Capítulo 2

Revisão bibliográfica

O objetivo proposto nesse trabalho, de elaborar um artefato didático para o ensino do tópico de *deep learning*, se baseia em alguns fundamentos teóricos e na relação com trabalho similares a esse que já tenham sido desenvolvidos. Este capítulo tem por objetivo elencar esses fundamentos e esses trabalhos correlatos, visando fundamentar as decisões tomadas na elaboração deste trabalho.

2.1 Fundamentação teórica

2.1.1 Processamento Digital de Imagens

A popularização de computadores com alto poder de processamento (sejam *desktops* ou os *smartphones* de última geração), os avanços das tecnologias de sensores coloridos e a aplicação de técnicas de processamento de imagens digitais em campos variados, com destaque especial para a área da biomedicina, tem impulsionado os estudos e o desenvolvimento de novas técnicas e novas metodologias na área acadêmica de processamento digital de imagens, gerando grande interesse na área [7].

Em geral, esse interesse deriva de duas grandes áreas de aplicação: a melhoria da informação pictorial para interpretação humana e o processamento de imagens para armazenamento, transmissão e representação para percepção autônoma (a chamada *visão computacional*). É possível ainda dividir esses dois segmentos em áreas de estudo mais específicos [8]:

- Aquisição: digitalizar uma imagem registrada de outras formas ou mesmo preparar uma imagem já digitalizada para futuro processamento;
- Realce: destacar ou corrigir detalhes, ou mesmo revelar as informações de interesse mais relevantes presentes na imagem;
- Restauração: é semelhante ao realce, mas é mais objetiva e usa modelos matemáticos e/ou probabilísticos para tratar imagens degradadas;
- Processamento de cores: se baseia nas informações de cores da imagem para corrigir ou extrair informações presentes;
- Wavelets e processamento multirresolução: uma abordagem matemática que divide a imagem em pequenas partes menores para fins de representação e compressão;

- Compressão: como o nome já diz, lida com técnicas para reduzir o espaço ocupado pela imagem, em equilíbrio com a qualidade da mesma;
- Morfologia: aborda a extração de formas dentro das imagens para fins de representação e descrição;
- Segmentação: particionamento de imagens em segmentos ou objetos relevantes para estudo;
- Representação e descrição: lida geralmente com o resultado da segmentação e aborda formas automáticas de interpretação e diferenciação dos elementos segmentados
- Reconhecimento: lida com a classificação de objetos baseada em seus descritores.

Paralelamente, esse interesse (e até mesmo a necessidade) pelo avanço do campo de processamento digital de imagens se dá pela presença cada vez mais forte dessas mesmas tecnologias em tarefas corriqueiras da vida cotidiana. Com o advento da internet sem fio de banda larga, mais e mais pessoas carregam em seus bolsos e bolsas dispositivos digitais de alta capacidade de captura, processamento e transmissão de imagens [9]. É impossível imaginar que uma simples videoconferência via aplicações como *Skype* ou *Facetime* seriam possíveis sem algum tipo de processamento (mais especificamente, de compressão) de imagens. A explosão de aplicativos de compartilhamento de fotos como *Instagram*, *WhatsApp* e *Snapchat* inundam a web diariamente com quantidades massivas de fotos processadas, tratadas, melhoradas e comprimidas com apenas alguns toques na tela do celular. Um estudo do *think tank* L2 reporta, entre outros números assustadores, que cerca de 55 milhões de fotos são compartilhadas diariamente via Instagram [10]. Ainda em termos de redes sociais, problemas como restrição de idade e de exposição pornográfica (e até de pedofilia) tem motivado essas grandes empresas a investirem amplamente em técnicas de reconhecimento e classificação automática de imagem.

2.1.2 Aprendizado de Máquina e *Deep learning*

Lady Ada Lovelace, colaboradora de pesquisas de Charles Babbage, disse certa vez sobre a *máquina analítica*, inventada por Babbage: "[A máquina analítica] não tem qualquer pretensão de originar qualquer coisa. Ela pode fazer o que quer que saibamos como orientá-la a fazer... Seu objetivo é nos assistir a tornar possível o que nós já conhecemos" [11]. Eventualmente, essa fala foi citada pelo pioneiro do campo de Inteligência Artificial, Alan Turing, como "a objeção de *Lady* Lovelace", em seu artigo seminal de 1950, "Computação e Inteligência", famoso por introduzir o *teste de Turing*, além de outros conceitos que viriam a moldar o que conhecemos como inteligência artificial. Turing estava citando Ada Lovelace ao refletir sobre a capacidade dos computadores de eventualmente aprenderem e criarem, e ele chegou a conclusão que sim [12].

O aprendizado de máquina surge dessa questão: um computador é capaz de ir além de "o que quer que saibamos como orientá-lo a fazer" e aprender por si só a como desempenhar uma tarefa específica? Um computador é capaz de nos surpreender? Ao invés de programadores construindo regras de processamento de dados à mão, poderia um computador aprender automaticamente essas regras, simplesmente olhando para os dados?

Foi esse tipo de questionamento que abriu as portas para um novo paradigma de programação. Na programação clássica, humanos criam regras (um programa) e submetem

dados para serem processados de acordo com essas regras, e o computador devolve respostas. Com o aprendizado de máquina, os humanos fornecem tanto os dados como as respostas esperadas a partir deles, e o computador fornece as regras. Essas regras podem ser aplicadas agora a novos dados, para produzir respostas originais. Assim, um sistema baseado em aprendizado de máquina é *treinado*, ao invés de *programado* (explicitamente). Ao ser apresentado a uma quantidade considerável de exemplos relevantes à tarefa, encontra estruturas estatísticas nesses exemplos que permitem eventualmente criar regras para automatizar essa tarefa [11].

Dentro do domínio do aprendizado de máquina, se desenvolveu uma área conhecida como *deep learning* (ou "aprendizado profundo"), que diz respeito a uma nova abordagem para a representação das estruturas de aprendizado de máquina que enfatiza o aprendizado de camadas de representações cada vez mais significativas. No *deep learning*, essas camadas de representação são (quase sempre) aprendidas por meio de um modelo conhecido como *redes neurais*. O termo "neural" é uma referência à neurobiologia mas, embora muito dos conceitos fundamentais do *deep learning* tenham sido desenvolvidos em parte pela inspiração no entendimento corrente do cérebro, os modelos de aprendizado profundo não são modelagens do cérebro humano, e não há qualquer evidência que o cérebro implemente qualquer coisa como os mecanismos usados nos modelos atuais de *deep learning* [11].

Embora os primeiros modelos desenvolvidos nessa área não sejam tão recentes, ela começou a obter mais proeminência no ambiente acadêmico do processamento de imagens no começo dos anos 2010, em grande parte pelo avanço da disponibilidade de grande poder computacional, em termos de velocidade de processamento e armazenamento de memória [2]. Com processadores potentes e vasta memória disponível, é possível usar computadores para criar redes neurais de múltiplas camadas (daí o nome de aprendizado *profundo*), e isso traz diversas vantagens:

- **Simplicidade:** Ao invés de criar soluções específicas e restritas, as redes neurais profundas oferecem uma arquitetura de blocos básicos, as camadas, que podem ser repetidas e conectadas de diversas formas, conforme for apropriado e desejado.
- **Escalabilidade:** Modelos de *deep learning* são facilmente escalados para volumes massivos de dados. Modelos mais antigos de inteligência artificial tendem a perder performance e qualidade com volumes muito grandes.
- **Transferência de domínio:** Um modelo aprendido para uma tarefa pode ser aplicável a outras tarefas relacionadas, e as características aprendidas tendem a ser genéricas o suficiente para trabalhar com uma série de tarefas nas quais os dados disponíveis são poucos.

Dado o tremendo sucesso das redes neurais profundas, as técnicas de *deep learning* atualmente são o estado da arte para detecção, segmentação, classificação e reconhecimento de imagens [2]. Como visto antes, esses são exatamente os pilares do processamento digital de imagens. Em geral, o *deep learning* tem alcançado sucesso em diversas áreas, todas elas historicamente difíceis para o aprendizado de máquina tradicional:

- Classificação de imagens em um nível quase humano;
- Reconhecimento de fala em um nível quase humano;

- Transcrição de escrita manuscrita em um nível quase humano;
- Tradução de linguagens;
- Assistentes digitais, como a *Siri* da *Apple* e a *Alexa* da *Amazon*;
- Direção automobilística em um nível quase humano;
- e Habilidade de responder perguntas em linguagem natural, entre outras.

Observação Uma vez que o produto final desse trabalho é um tutorial que visa ensinar os conceitos introdutórios do *deep learning*, a maior parte do referencial teórico para esse assunto se encontra no próprio corpo do tutorial desenvolvido. Isso se dá pelo o intuito de tornar a experiência de aprendizagem por meio desse artefato, por parte do aluno, o mais independente e autocontida possível, isso é, que tudo que ele precise para percorrer o tutorial esteja presente nele mesmo, sem a necessidade de consulta a fontes externas como livros, artigos e até mesmo essa seção deste trabalho. Dessa forma, buscou-se apresentar aqui, nesse tópico a respeito de aprendizagem de máquina e *deep learning*, apenas uma breve introdução aos conceitos e sua relevância, de forma a justificar e embasar a escolha desses assuntos para o tutorial.

2.1.3 Aprendizado Baseado em Problemas

O documento das Diretrizes Curriculares Nacionais aprovadas pelo MEC para os cursos de graduação na área de computação destaca o trabalho da professora Jeannette Wing, da Carnegie Mellon University, a respeito do chamado *pensamento computacional*. Os trabalhos de Wing discutem "como o conhecimento sobre os métodos e modelos da Computação podem ser úteis à formulação e resolução de problemas do cotidiano, sendo fundamental ao rol de habilidades do cidadão do Século XXI" [6]. Desde então, cresceram as pesquisas e iniciativas sobre o desenvolvimento do pensamento/raciocínio computacional na Educação e o importante papel do licenciado em Computação nesse contexto. Espera-se do profissional docente formado em Licenciatura a capacidade de fomentar habilidades de aprendizagem contínua, resolução de problemas, visão integrada do conhecimento e capacidade de aplicar a teoria relacionando-a à prática.

Uma metodologia que se tornou referência para a aplicação dessas exigências é a *Aprendizagem Baseada em Problemas* (PBL, *Problem Based Learning*) [13]. O PBL é uma abordagem de aprendizado ativo, centrada em um problema a ser resolvido e na busca criativa e independente do aluno de sua solução. O PBL difere das abordagens mais "tradicionais" em que os participantes são encorajados a utilizarem estratégias de aprendizado autocentradas, enfatizando a habilidade do indivíduo de buscar e assimilar informações relevantes para atacar o problema em mãos, aspectos fundamentais do pensamento computacional de Wing. Outro aspecto relevante do PBL é a ausência da figura do professor, uma vez que o foco está no *problema a ser resolvido*, não no *assunto a ser aprendido*, como nas abordagens tradicionais. O condutor de uma atividade PBL age muito mais como um *facilitador* do que como um *professor*, provendo encorajamento e orientação aos participantes.

A teoria do PBL foi o grande motivador desse trabalho quanto ao artefato a ser produzido, um tutorial baseado em um problema. Nesse tutorial, o aluno será apresentado

às ferramentas do *deep learning* e à teoria por trás dele para poder, de forma autônoma, independente e autodidata, resolver o problema da transferência de estilo artístico.

2.1.4 O formato tutorial

Quanto à forma do tutorial que será desenvolvido, foram tomadas duas decisões principais: o uso do *Jupyter Notebook* como ferramenta de apresentação e a escolha da aplicação da transferência de estilo artístico como objetivo final do estudo dirigido.

Jupyter Notebooks

Pesquisadores de todo tipo de disciplinas acadêmicas produzem códigos de programação que acompanham, fundamentam, exemplificam e aplicam seus trabalhos: obtenção e processamento de dados, testes e modelagens estatísticas, simulações e representações gráficas. Para tal, existem diversas bibliotecas e ferramentas disponibilizadas em código aberto (NumPy, Julia ou FEniCS, por exemplo), mas o código desenvolvido por esses pesquisadores, propriamente dito, para trabalhos específicos, costuma não ser publicado, reduzindo a reprodutibilidade dos resultados. A grande maioria dos autores opta por descrever a metodologia computacional por meio de prosa, como parte de uma descrição genérica da metodologia de pesquisa. Mas é sabido que a linguagem humana não tem a mesma precisão descritiva do código-fonte, e reproduzir esses métodos costuma não ser tão fácil ou confiável quanto deveria. Alguns ainda tentam prover seus códigos de forma separada, como material complementar, mas isso pode dificultar a relação entre a prosa e o código por parte do leitor, havendo ainda o risco do surgimento de inconsistências entre as partes, conforme o autor trabalha em ambos separadamente[1]. Assim, os chamados *notebooks*, documentos que integram prosa, código e resultados, oferecem uma forma de publicar um método computacional que pode ser rapidamente lido e replicado [14].

Notebooks são projetados para demonstrar o fluxo de trabalho científico-computacional, da exploração interativa à publicação dos resultados de um processo. O código em um *notebook* é organizado em células, partes que podem ser modificadas e executadas individualmente. O resultado de cada célula é exibido diretamente abaixo da mesma e salvo como parte do documento. Isso nada mais é do que a evolução dos *prompts* de comandos interativos, os REPLs[2], que foram por muito tempo a base da programação interativa [15]. Entretanto, se o resultado direto dos antigos *prompts* eram apenas texto, os *notebooks* permitem incluir saídas como gráficos, imagens, equações matemáticas formatadas e até mesmo controles interativos. O texto em prosa é alternado com o código e essa dinâmica forma uma narrativa computacional dinâmica e atrativa para o usuário.

O uso de *notebooks* se tornou popular primeiramente entre os matemáticos, e sistemas computacionais proprietários de álgebra, como o Mathematica e o Maple, usam esse tipo de interface, assim como o SageMath, de código-fonte aberto. O Jupyter, que será utilizado nesse trabalho, tem por objetivo alcançar uma audiência mais ampla. O extitJupyeré um projeto de código-fonte aberto que pode trabalhar diversas linguagens de programa-

[1] Sobre a relação entre a apresentação visual de um conteúdo e o processo de aprendizagem, mais no próximo tópico.

[2] Read-Evaluate-Print-Loop - laço de repetição de leitura, avaliação e impressão de resultado

ção diferentes. Utilizando um protocolo em comum e bem documentado, o extitJupyerse comunica com servidores de diferentes linguagens; mais de 50 já foram escritos.

O extitJupyerNotebook é acessado por meio de um navegador de internet. Isso torna prático usar a mesma interface rodando localmente como uma aplicação de *desktop* ou rodando em um servidor remoto. No último caso, a única aplicação que o usuário precisa é de um navegador; assim, por exemplo, um professor pode instalar a aplicação em um servidor e dar acesso a ela para diversos alunos facilmente. Os arquivos de *notebooks* são criados de forma simples, usando o formato JSON e extensão '.ipynb'.

Sendo o extitJupyeruma plataforma nascida e criada no ambiente da linguagem de programação Python, foi natural escolher utilizar essa mesma linguagem para o desenvolvimento do tutorial. Python tem uma série de vantagens que a tem feito a linguagem natural para aplicações de ciência de dados e suas aplicações no campo da visão computacional [3].

Estética e aprendizagem

Uma vez que o produto final deste trabalho é artefato educacional voltado para o estudo autodidata, viu-se necessário escolher um tema que atraia a atenção e o foco do aluno que irá utilizá-lo. Embora muito já tenha sido escrito sobre a importância da teoria a ser ensinada, a relevância e até mesmo a capacidade de entretenimento de um tutorial tal como esse são fundamentais para o sucesso na tentativa de motivar o estudante a dedicar seu tempo no acompanhamento e desenvolvimento das atividades propostas. Assim, dentre as possíveis aplicações da teoria de *deep learning*, escolheu-se a de transferência de estilo artístico, em que se busca combinar em uma imagem final o conteúdo de uma imagem inicial qualquer com os padrões visuais estéticos de outra.

Ainda é comum atualmente, principalmente na área das ciências exatas, a noção de que emoção e cognição são funções humanas independentes. Por muito tempo tomou-se como certo que o cognitivo e o emocional operavam como dois sistemas separados, partindo do princípio que emoções e sentimentos eram funções de baixa ordem, se comparadas às funções mais elevadas do pensamento racional [16].

Pesquisas atuais tem derrubado quase que completamente essas concepções. É sabido que emoção e cognição são interdependentes: emoções são cruciais para avaliar rapidamente uma situação e para o processo de tomada de decisão. Dados mostram como emoções positivas são particularmente importantes para aprendizagem, solução de problemas, trabalho criativo e inovação, em grande parte porque permitem e promovem um pensamento mais flexivo e adaptado às circunstâncias [17].

Embora as praticas mais comuns do design instrucional foquem nos aspectos cognitivos do aprendizado, novas linhas de pequisa tem explorado o papel da dimensão afetiva nesse processo. O campo conhecido como "design emocional" busca verificar as formas nas quais o humor e os sentimentos do aprendiz podem influenciar sua motivação na, e os resultados da, aprendizagem.

Uma das formas mais óbvias de influenciar a dimensão afetiva de alguém é por meio da estética visual. Assim, a importância do *design* no ensino tem ganhado força e se tornado cada vez mais importante nos últimos anos. Pesquisas iniciais já tem mostrado como evocar emoções positivas nos alunos por meio de uma experiência visual atrativa (cores, imagens, *layouts*, etc) pode facilitar uma experiência de aprendizado bem sucedida [18].

Isso significa que materiais educacionais visualmente atrativos tem um grande potencial de motivar e enriquecer o processo de aprendizagem.

Em seu livro sobre soluções visuais para problemas de aprendizagem e ensino, Connie Malamed lista quatro formas principais pelas quais a estética pode evocar emoções positivas que, por sua vez, facilitam o aprendizado:

Juízo de valor - As pessoas fazem juízos rápidos baseados nas experiências sensoriais e são instantaneamente atraídos para objetos esteticamente agradáveis, enquanto rejeitam o que não o são. Alunos tendem a julgar os materiais que utilizam em seu estudo da mesma forma. Materiais visualmente agradáveis não só aumentam o apelo da experiência de aprendizagem, como também a probabilidade do aluno julgar o valor e a credibilidade do próprio. E o valor atribuído ao material é um fator primordial para esse processo.

Maior motivação - Quando alunos experimentam emoções positivas causadas por uma estética agradável, isso pode fomentar a motivação ou o desejo intrínsecos de aprender [19]. Além disso, em uma estética atraente, as tarefas de aprendizagem tendem a ser percebidas como mais fáceis, quando comparadas a um *design* neutro [20].

Fuga de emoções negativas - Um fator associado à estética é a usabilidade, ou a facilidade com que se chega ao objetivo pretendido. Um *design* esteticamente atrativo terá a simplicidade requerida para tal. Isso ajuda o usuário a evitar emoções negativas associadas a um *design* ruim, como frustração e insatisfação, enriquecendo assim o aprendizado.

Facilidade de uso percebida - As pessoas julgam que objetos esteticamente agradáveis são mais amigáveis e fáceis de usar [21]. Em um experimento com estações de autoatendimento bancárias, os sujeitos afirmaram que as telas mais agradáveis eram mais fáceis de usar do que as telas neutras, mesmo que ambas apresentassem os mesmos botões e funcionalidades. A experiência do usuário foi enriquecida pela estética visual.

Por mais que se trate de uma área, por definição, visual, a visão computacional tem seu fundamento teórico na matemática e, por isso, por vezes seu ensino é uma experiência frustrante para os alunos. Considerando essa forte influência da estética nos processos de interação e apreensão do mundo externo, não é de se surpreender que ela tenha um impacto significativo na cognição e no aprendizado, e isso tornou quase automática a decisão de utilizar uma aplicação com forte apelo visual e estético no desenvolvimento deste trabalho.

2.2 Trabalhos correlatos

Ao iniciar o desenvolvimento desse trabalho, ainda na fase de levantamento de referencial bibliográfico, alguns trabalhos relacionados ao proposto aqui foram encontrados e serviram de exemplo e inspiração para o tutorial que foi desenvolvido, e estão listados aqui.

2.2.1 Curso aberto de *deep learning* da Universidade de Stanford

A *CS231n - Convolutional Neural Networks for Visual Recognition* (CS231n - Redes Neurais Convolucionais para Reconhecimento Visual) é uma disciplina da Universidade de Stanford que tem sido ministrada desde 2005, idealizada pela Dra. Li Fei-Fei, e se tornou o principal referencial na internet para o ensino de *deep learning* desde que suas

notas começaram a ser divulgadas publicamente [22]. O CS231n é um curso de dez semanas que visa ensinar os alunos a implementar, treinar e depurar suas próprias redes neurais com enfoque na classificação de imagens, um dos problemas mais fundamentais da visão computacional. *A maior parte da teoria de deep learning apresentada no tutorial produzido neste trabalho, assim como a estrutura do conteúdo e a progressão dos assuntos, foi baseada na estrutura do CS231n, toda disponível gratuitamente em seu site oficial.*

2.2.2 *Um algoritmo neural de estilo artístico*

Um artigo científico publicado em 2015 tomou de assalto a comunidade acadêmica da área de visão computacional. Nesse artigo, *A Neural Algorithm of Artistic Style* [23] (Um Algoritmo Neural de Estilo Artístico), Gatys *et al.* explicam como modelos de *deep learning* podem ser usados muito eficazmente para extrair características perceptivas relacionadas ao estilo artístico de uma imagem, tais como cores, técnicas de pintura e texturas de tintas e superfícies. Em particular, esse artigo fala sobre o que se chama na matemática de *problema de otimização*. Esse problema está por trás da técnica desenvolvida por Gatys *et al.* de não só extrair essas características de uma determinada imagem, mas também aplicá-la a outra imagem, resultando em uma foto qualquer aparentando ter sido pintada por Vincent Van Gogh ou Cândido Portinari. *O forte apelo visual estético da técnica desenvolvida por Gatys et al. e a fundamentação matemática do algoritmo que a implementa forneceram a inspiração para criar um tutorial que ensine justamente essa técnica e apresentam exatamente o que foi dito anteriormente sobre a força da estética como motivação para o processo de ensino e aprendizagem. O aluno se interessa pelo apelo estético e chega do outro lado do tutorial tendo aprendido a matemática e a computação por trás de uma técnica de deep learning estado da arte.*

2.2.3 Tutoriais interativos do *Keras.js*

O Keras é uma biblioteca de *deep learning* para o *Python* [24]. Como quase tudo que é desenvolvido atualmente em termos de modelos e paradigmas de programação, existe uma adaptação do *Keras* para a linguagem *javascript*, o *Keras.js*[3], que permite executar e analisar modelos desenvolvidos no *Keras* diretamente no navegador.

O mais interessante e chamativo do ambiente do *Keras.js* é a possibilidade de acompanhar os resultados de todos os passos e todas as camadas de representações que compõe uma rede neural profunda. O que era apenas um conceito abstrato, de características perceptivas de uma imagem representada em camadas de neurônios artificiais interligados é mostrado como uma série de imagens e valores que tornam quase que palpável a transformação de uma imagem em traços, gráficos e vetores, até se transformar em um único número, que representa a porcentagem de confiança do modelo de que o número que o usuário rabiscou com seu *mouse* em um pequeno quadrado branco no início da tela seja um número 3 (ou qualquer outro que tenha sido rabiscado, com porcentagem de acerto de quase 99%!). *Esse aspectos interativos e significantes do Keras.js, isto é, de mostrar em tempo real e de forma simplificada para o usuário o sentido por trás da matemática e das estruturas computacionais que fundamentam e compõe o deep learning inspirou toda a decisão pela forma do tutorial a ser desenvolvido, com trechos de teoria alternados com*

[3] https://transcranial.github.io/keras-js/

exercícios práticos de resultado imediato e representações gráficas de praticamente tudo que é discutido.

Uma vez fundamentadas as decisões tomadas para a elaboração do tutorial de transferência de estilo artístico para o ensino de *deep learning*, o próximo capítulo apresenta o tutorial propriamente dito, conforme ele será apresentado para o aluno.

Capítulo 3

Solução proposta

Uma vez estabelecidos os fundamentos teóricos que embasam as decisões tomadas no presente trabalho no capítulo anterior, apresenta-se agora o artefato final, o tutorial de uso de *machine learning* para realizar a transferência de estilo artístico.

O tutorial será reproduzido aqui em sua totalidade, e esse mesmo conteúdo será disponibilizado gratuitamente na internet na forma de *Jupyter Notebooks*, acessíveis por meio de um repositório aberto de código[1].

3.1 Introdução

Um aplicativo para celulares chamado Prisma[2] recentemente se tornou muito popular entre os usuários das redes sociais[3]. O grande atrativo do Prisma é sua capacidade de transformar fotos pessoais em obras de arte, utilizando filtros baseados em obras famosas, aplicando estilos e estética dessas obras às fotos do usuário. Esse aplicativo realiza esse tipo de operação, chamada de *transferência de estilo*, com a ajuda de um ramo da área de aprendizado de máquina chamado de *redes neurais convolucionais* (RNCs)[4]. O objetivo desse tutorial é iniciar uma jornada pelo mundo das redes neurais convolucionais, da teoria à prática, ao tentarmos reproduzir sistematicamente os efeitos visuais do Prisma.

3.1.1 O que é o Prisma e como ele funciona

O Prisma é um aplicativo *mobile* que te permite transferir o *estilo* de uma imagem, como um mosaico *art nouveau* inspirado em Alphonse Mucha [26], por exemplo, para o **conteúdo** de outra, como uma paisagem da sua cidade [27], alcançando resultados visuais atraentes (Figura 3.1).

O resultado da aplicação do filtro do Prisma é agradável para muitas pessoas, e o objetivo desse tutorial é descobrir a ciência por trás dessa técnica. Para quem estudou um pouco de Processamento Digital de Imagens, à primeira vista, as primeiras ideias a respeito desse "como" podem envolver a extração de características de baixo nível como

[1] https://gitlab.com/fschulz88/tutorial-deep-learning
[2] https://prisma-ai.com/
[3] *Intelligent photo effects app taps into deep learning for an edgy art connection* [25]
[4] https://prismalabs.ai/about.html

Figura 3.1: À esquerda, a imagem **c** de onde vem o conteúdo, a imagem **s**, de onde vem o estilo e, à direita, a imagem **x**, com o estilo transferido, gerada pelo Prisma

cor e textura da *imagem de estilo* **s**, e aplicá-las às características mais semânticas, de alto nível, da imagem da paisagem de Londres (**c**), para chegar na imagem **x**

A pergunta é, como sequer começar a desenvolver uma aplicação que faça isso? Talvez alguma análise *pixel*-a-*pixel* na imagem **s** possa obter coisas como média de cores ou alguns aspectos de textura. Mas isso leva a uma pergunta mais complicada ainda: como "explicar" para a aplicação que a textura e as cores que estamos interessados estão no *nível das pinceladas* de uma imagem, mas não nas *formas gerais* da pintura?

Digamos que fôssemos por esse caminho. Como então *aplicar seletivamente* esse estilo sobre o conteúdo de **c**? Não há como copiar e colar esse estilo sem perder aspectos essenciais da estrutura do conteúdo. Nessa mesma linha, como descartar de forma não prejudicial o estilo já presente na imagem **c**?

Assim como o Prisma tomou de assalto as redes sociais, alguns anos antes de seu lançamento um artigo científico publicado em 2015 se tornou muito popular entre a comunidade acadêmica da área de visão computacional. Nesse artigo[5], vemos *exatamente* a explicação para como o Prisma funciona. Em particular, esse artigo fala sobre o que se chama na matemática de *problema de otimização*.

Digamos que haja uma maneira de medir o quão *diferentes em conteúdo* duas imagens são entre si. Isso significa que teríamos uma função que tende a **0** quando as duas imagens de entrada (**c** e **x**) são muito próximas entre si em termos de conteúdo, e cresce conforme o conteúdo delas diverge. Chamamos essa função de *perda de conteúdo*:

$$\mathcal{L}_{\text{conteúdo}}\left(\quad , \quad \right) \approx 0$$

Figura 3.2: Um esboço da função de *perda de conteúdo*

Digamos também que haja uma outra função que nos diga o quão *similares em estilo* duas imagens são entre si. Novamente, essa função cresce conforme as duas imagens de entrada (**s** e **x**) tendem a divergir em estilo. Chamamos essa função de *perda de estilo*:

[5] *A Neural Algorithm of Artistic Style* [23]

$$\mathcal{L}_{\text{estilo}}\left(\ \ ,\ \ \right) \approx 0$$

Figura 3.3: Um esboço da função de *perda de estilo*

Se tivéssemos essas duas funções, então o problema da transferência de estilo seria fácil, certo? Tudo que precisaríamos seria encontrar uma imagem $\mathbf{x}$ que diferisse o mínimo possível *em termos de conteúdo* da imagem de conteúdo $\mathbf{c}$ ao mesmo tempo em que diferisse o mínimo possível *em termos de estilo* da imagem de estilo $\mathbf{s}$. Em outras palavras, nos gostaríamos de minimizar simultaneamente tanto a perda de estilo quanto a de conteúdo.

Podemos representar isso matematicamente de forma levemente assustadora da seguinte forma:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \left(\alpha \mathcal{L}_{\text{conteudo}}(\mathbf{c}, \mathbf{x}) + \beta \mathcal{L}_{\text{estilo}}(\mathbf{s}, \mathbf{x})\right)$$

Aqui, α e β são apenas números, parâmetros, que nos permitem controlar o quanto queremos enfatizar do conteúdo em relação ao estilo. Veremos mais adiante alguns efeitos de mexer nos pesos desses fatores.

A ideia crucial introduzida pelo artigo de Gatys [23] é que as definições dessas perdas de conteúdo e estilo não são baseadas em *diferenças pixel-a-pixel* entre as imagens mas, ao invés disso, em diferenças *perceptivas de mais alto-nível* entre elas. Ótimo, mas como que se escreve um algoritmo que entende o suficiente do *significado* de uma imagem para perceber essas diferenças semânticas?

A verdade é que isso é impossível. Pelo menos no sentido clássico de um algoritmo como um conjunto fixo de instruções. Para tal, precisamos nos voltar para o aprendizado de máquina, que é uma excelente ferramenta para resolver problemas genéricos como esses, que parecem até intuitivos, mas que são difíceis de estruturar explicitamente em passos para encontrar a solução. Ao longo desse tutorial, queremos mostrar exatamente como fazer isso.

Começamos de um ponto relativamente básico: *o problema da classificação de imagens*. Vamos caminhar de forma bem suave pelas soluções para esse problema até estarmos familiarizados com uma área do *machine learning* que é ótima para lidar com imagens, chamada *redes neurais convolucionais* (as *RNCs*). Veremos como as RNCs podem ser usadas para definir essas funções de perdas perceptivas centrais ao nosso problema de otimização de transferência de conteúdo. Vamos concluir com uma implementação concreta da solução do problema com a qual você poderá experimentar e expandir.

Nosso objetivo é começar em um ponto relativamente básico e acessível e gradualmente aumentar a complexidade conforme avançamos, para que você aprenda algo interessante, não importa o seu nível de conhecimento prévio a respeito de *machine learning*.

3.2 O problema da classificação de imagens

Essa seção é baseada em um curso à distância da Universidade de Stanford, Convolutional Neural Networks for Visual Recognition (CS231n). Caso você tenha interesse no

assunto, o curso é gratuito: `http://cs231n.stanford.edu/`.

Começamos nossa jornada a partir do problema da *classificação de imagens*. Esse é um problema enganosamente simples de propor: dada uma imagem de entrada, faça com que um computador automaticamente classifique-a como parte de um conjunto fixo de categorias, tais como "bebê", "cachorro", "carro" ou "escova de dentes". E a razão pela qual estamos começando com esse problema é que ele está na base de várias tarefas da visão computacional que parecem não ter relação entre si, incluindo a busca por reproduzir o efeito visual do Prisma.

Em termos mais precisos, imagine uma imagem com três níveis de cor (RGB) que tenha **L** *pixels* de largura e **A** *pixels* de altura. Essa imagem pode ser representada em um computador como um vetor de $D = L \times A \times 3$ números, cada um indo de **0** (mínimo de luminosidade) a **1** (luminosidade máxima). Vamos assumir também que nós temos **K** categorias de coisas em que gostaríamos de classificar imagens. A tarefa então passa a ser a de encontrar uma função que receba como entrada um desses *arrays* de números e imprima como saída o rótulo correto do nosso conjunto de categorias, como "bebê", por exemplo.

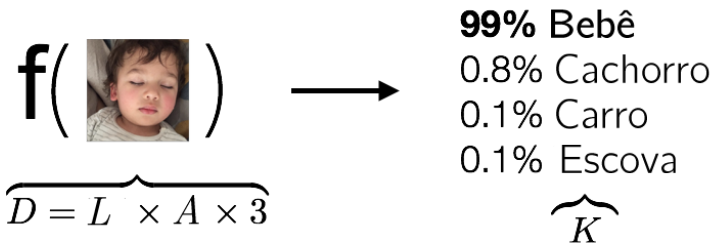


Figura 3.4: O problema de classificação de imagens

Na verdade, ao invés de apenas imprimir o nome de uma categoria, seria mais útil obter um *valor de confiança* para cada categoria. Assim, nós não teríamos apenas a categoria principal pela qual estamos buscando (o maior valor), mas também teríamos uma ideia de quanta confiança nós temos na nossa classificação. Assim, em essência, o que estamos buscando é uma *função de confiança* $\mathbf{f} : \mathbb{R}^D \mapsto \mathbb{R}^K$ que mapeia a representação em dados de uma imagem a valores de categorias de classificação.

Como poderíamos escrever uma função assim? Uma abordagem mais ingênua seria codificar diretamente algumas características de bebês (tais como cabeças grandes, nariz escorrendo, bochechas fofas, etc) na função. Mas mesmo que você soubesse como fazer isso, e se você quisesse classificar carros? E diferentes tipos de carros? E escovas de dentes? E se o conjunto **K** de categorias se tornasse grande e arbitrário?

Para complicar mais ainda o problema, note que qualquer mudança, por menor que seja, no contexto no qual a imagem foi capturada (iluminação, ponto de vista, plano de fundo, etc), afeta bastante os números que representam a imagem que seriam passados como entrada para a nossa função. Como escrever uma função de classificação que ignore esse tipo de diferença supérflua e, ao mesmo tempo, tenha a habilidade de distinguir entre um "bebê" e uma "criança"?

Essas questões tem a mesma ordem de dificuldade que definir *diferenças perceptivas* para o problema de transferência de estilo que vimos antes. E a razão para isso é que há uma *diferença semântica* entre a representação das imagens na entrada (um *array*

Figura 3.5: Alguns dos desafios da classificação de imagens por computador [22].

de números) e o que estamos buscando (uma classificação de categorias). Assim, vamos deixar de lado a tentativa de escrever essa função de classificação nós mesmos, e vamos deixar que o *machine learning descubra automaticamente as representações apropriadas* necessárias para resolver para nós esse problema.

Esse conceito é a base intelectual de todo esse projeto.

3.2.1 Uma abordagem de aprendizado supervisionado para o problema de classificação de imagens

O ramo do *machine learning* que vamos utilizar para resolver o problema da classificação de imagens é chamado de *aprendizado supervisionado*. Quando você ouve a maioria das pessoas falando de *machine learning* hoje, seja o *deep learning* ou outra coisa, o que provavelmente eles estarão se referindo é ao aprendizado supervisionado, que é o subconjunto do *machine learning* que alcançou mais sucesso nos últimos anos. Aprendizado supervisionado se tornou o procedimento padrão para aprender com dados, e está esboçado a seguir no contexto do problema de classificação de imagens (Figura 3.6).

- Começamos com um conjunto de imagens de exemplo pré-classificadas, o que significa que temos um conjunto de imagens com rótulos conhecidos. Isso é chamado de *dados de treinamento*, e é a base com a qual nosso sistema vai aprender.
- A função que queremos encontrar é chamada de *função de valor*, que mapeia uma dada imagem para valores de confiança de categorias. Para definir o que estamos procurando, primeiro damos um palpite para sua forma funcional e fazemos com que ela varie conforme diversos *parâmetros* θ que precisamos encontrar.
- Introduzimos a *função de perda* $\mathcal{L}$, que quantifica a variância entre o que a nossa função de valor sugere para os valores de categoria e o que os nossos dados de treinamento informam como sendo a verdade conhecida. Assim, a função de perda aumenta se a função de valor está fazendo um trabalho ruim e diminui se estiver indo bem.
- Finalmente, nós temos um *algoritmo de otimização* ou *aprendizado*. Ele é um mecanismo para alimentar o nosso sistema com um monte de dados de treinamento e fazê-lo aumentar iterativamente a função de valor ao mudar os valores dos parâme-

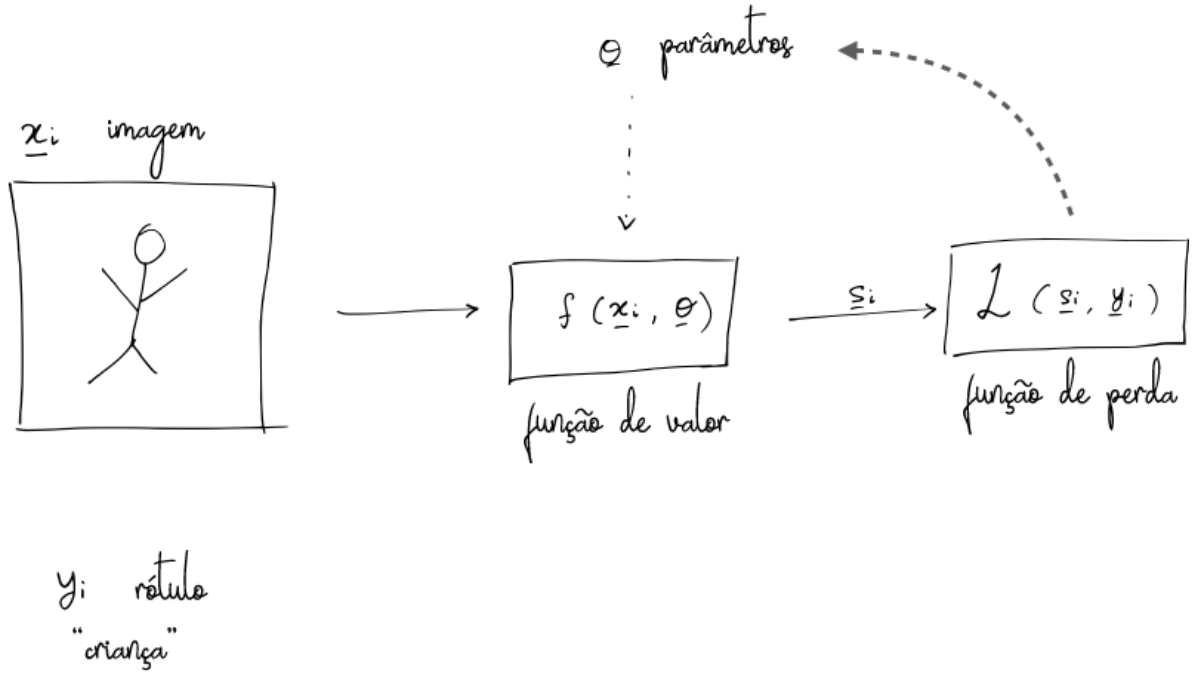


Figura 3.6: As peças que compõe a solução de aprendizado supervisionado para o problema de classificação de imagens.

tros θ. O objetivo do processo de aprendizado é determinar os parâmetros que nos dão a melhor (isso é, a menor) perda.

Uma vez que completamos esse processo e aprendemos uma função de valor apropriada, esperamos que ela possa ser *generalizada*. Isso é, uma função que funcione bem para uma entrada genérica (chamada de *dados de teste*) que nunca tenham sido vistos antes.

Muito da empolgação com as técnicas de *machine learning* hoje vem de fazer escolhas específicas para essas partes diferentes, permitindo que possamos aprender funções poderosas que consigam mapear conjuntos diversos de entradas e saídas. A seguir, vamos fazer escolhas cada vez mais sofisticadas para essas peças com o objetivo de implementar soluções cada vez melhores para o problema de classificação de imagens.

Um classificador de imagens linear

Lembrando: o problema que estamos tentando resolver é a classificação de imagens. Temos uma imagem $\mathbf{x}$ que é representada por um vetor $D = (L \times A \times p$, onde p, a *profundidade de cores*, é 1 para imagens em escala de cinza e 3 para imagens coloridas em RGB). O que queremos é descobrir a qual categoria (em um conjunto de $\mathbf{K}$ categorias) ela pertence. Assim, estamos interessados em encontrar uma *função de valor* $\mathbf{f} : \mathbb{R}^D \mapsto \mathbb{R}^K$ que mapeie dados de imagem a valores de classe.

O exemplo mais simples de uma função assim é um mapeamento linear:

$$\mathbf{f}(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b},$$

que introduz dois parâmetros que precisamos entender: a matriz $\mathbf{W}$ (de tamanho $K \times D$, que chamamos de *pesos*), e o vetor $\mathbf{b}$ (de tamanho $K \times 1$, que chamamos de *tendências*).

Uma vez que queremos interpretar o vetor de saída desse mapeamento linear como a probabilidade de cada categoria, nós passamos esse resultado por algo que chamamos de *função softmax* σ, que normaliza os valores para um conjunto de números entre 0 e 1 cuja soma é 1.

$$s_j = \sigma(\mathbf{f})_j = \frac{e^{f_j}}{\sum_{k=1}^{K} e^{f_k}}$$

Suponhamos que os nossos dados de treinamento são um conjunto de N exemplos pré-classificados $\mathbf{x_i} \in \mathbb{R}^D$, cada um com a categoria correta $y_i \in 1, \ldots, K$. Uma boa forma funcional para determinar a perda total em todos esses exemplos é a *perda de entropia cruzada* [28]:

$$\mathcal{L}(\mathbf{s}) = -\sum_i log(s_{y_i})$$

Para cada exemplo, essa função simplesmente pega o valor da classe para a classe correta e tira seu log negativo. Dessa forma, quanto mais perto de 1 for esse valor (ou seja, está correto), mais perto de 0 será a perda, e quanto mais perto de 0 estiver o valor (está errado), maior a perda. A perda de entropia cruzada serve então como uma função que quantifica o quão ruim a função de valor se comporta para dados conhecidos.

Agora que nós temos uma função de perda que mede a qualidade do nosso classificador, tudo que precisamos fazer é encontrar os parâmetros (pesos e tendências) que minimizem essa perda. Esse é um *problema de otimização* clássico e, no exemplo prático que iremos ver a seguir, vamos usar para isso um método chamado *gradiente descendente estocástico* [28].

Para ter uma noção de como esse método funciona, vamos pensar em uma função de perda mais simples que tenha apenas um parâmetro, w. Essa função se parece com a curva na Figura 3.7, e estamos tentando encontrar o w_{otimo}, onde a perda $\mathcal{L}(w)$ está no ponto mais baixo da curva.

Uma vez que nós não sabemos onde começar com isso, começamos com um palpite em um ponto w_0 qualquer. A partir daí, exploramos as proximidades e vamos na direção descendente da curva. Isso nos leva ao ponto w_1. Nele, fazemos a mesma coisa novamente, chegando em w_2, e assim por diante, até chegamos um ponto em que não houver mais para onde descer, ou a curva for 0. É aí que está o nosso w_{otimo}

Isso é mostrado nas equações ao longo da Figura 3.7, onde agora introduzimos um parâmetro η, chamado de *taxa de aprendizado*. Essa é uma medida do quão rápido nós modificamos nossos pesos e é algo que precisamos afinar com cuidado na prática.

O mecanismo de fato para encontrar esses pontos na curva (ou gradientes, se tivermos múltiplos parâmetros) vem por padrão na biblioteca *TensorFlow*, como veremos a seguir. Caso você tenha interesse nos detalhes de como isso é feito, procure ler sobre algo chamado de *modo reverso de diferenciação automática* [29].

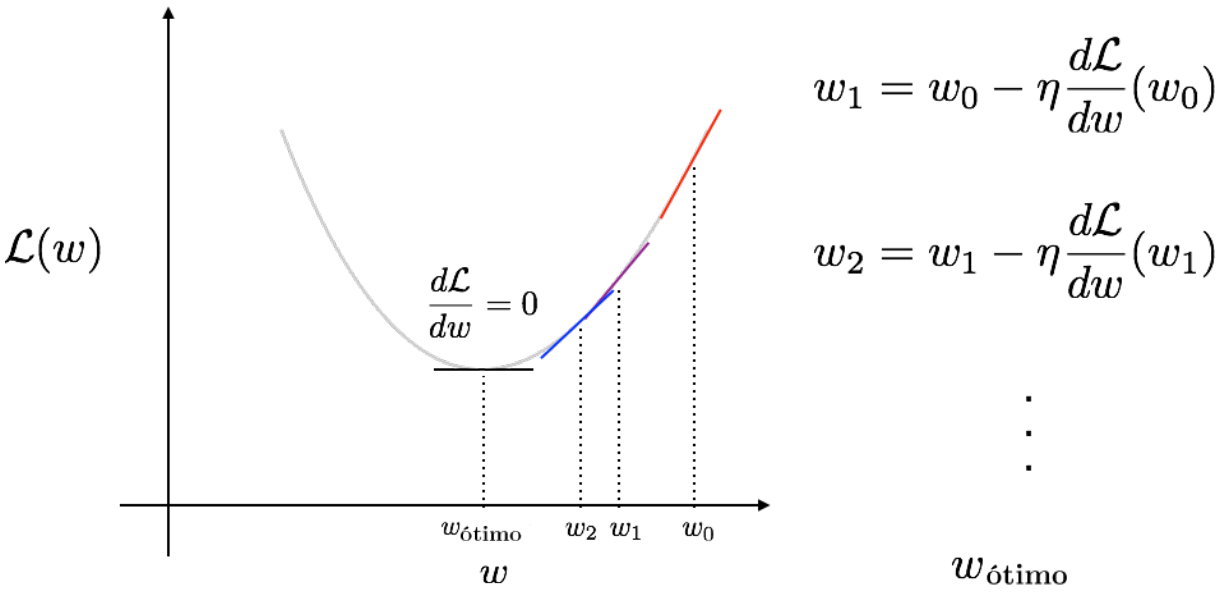


Figura 3.7: Uma versão simplificada do *gradiente descendente*

Exercício 1: Um classificador linear para o conjunto de algarismos MNIST

Finalmente temos todas as peças no lugar para construirmos nosso primeiro classificador de imagens! Dadas algumas imagens como um vetor de números, temos uma função de valor parametrizada (transformação linear seguida de uma função *softmax*) que nos leva a valores de categoria. Temos uma forma de avaliar a performance dessa função (a função de perda de entropia cruzada). Também temos um algoritmo para aprender com dados de exemplo e melhorar os parâmetros do classificador (a otimização por meio do gradiente descendente estocástico).

O exercício a seguir é baseado em um exemplo do site da biblioteca *TensorFlow* [30] e nos permitirá codificar um classificador de imagens no qual usaremos todas essas peças que falamos a respeito na prática.

Começamos importando alguns pacotes necessários:

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (5.0, 4.0)
plt.rcParams['image.cmap'] = 'Greys'

import numpy as np
np.set_printoptions(suppress=True)
np.set_printoptions(precision=2)
```

O MNIST [31] é um conjunto de dados que contem 70.000 imagens rotuladas de algarismos escritos à mão, como na Figura 3.8

Nós vamos treinar um classificador linear em uma parte desse conjunto, e testá-lo em relação a outra parte do conjunto para saber como foi nosso desempenho.

```
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

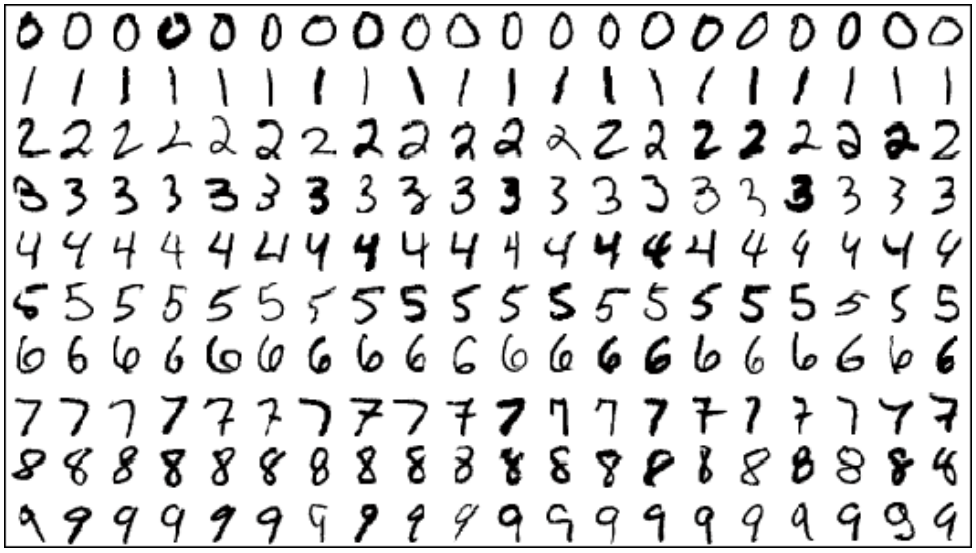


Figura 3.8: Alguns exemplos do conjunto de imagens no MNIST

O conjunto do MNIST já vem dividido em três partes:

- Um conjunto (55.000 exemplos) de treinamento
- Um conjunto de validação (5.000 exemplos) que pode ser usado para otimizar parâmetros
- Um conjunto de teste (10.000 exemplos) usados para medir a acurácia do modelo treinado

As imagens estão em escala de cinza e tem 28 *pixels* de largura por 28 de altura, armazenadas em um vetor de largura 784.

Os rótulos estão em um vetor de largura 10, o que significa um vetor só de zeros, exceto pela localização que corresponde ao rótulo ao qual ele se refere. Por exemplo, uma imagem com o rótulo **3** é representada como $(\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0})$.

O código

```
print mnist.train.images.shape
print mnist.train.labels.shape
```

retorna

```
(55000, 784)
(55000, 10)
```

e o código

```
print mnist.test.images.shape
print mnist.test.labels.shape
```

retorna

```
(10000, 784)
(10000, 10)
```

Isso fica mais claro quando visualizamos uma imagem e seu respectivo rótulo. O código

```
example_image = mnist.train.images[1]
example_image_reshaped = example_image.reshape((28, 28)) # Can't render
    a line of 784 numbers
example_label = mnist.train.labels[1]

print example_label
plt.imshow(example_image_reshaped)
```

retorna o vetor

```
[ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
```

e a imagem da Figura 3.9.

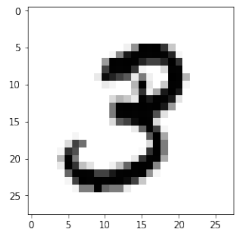


Figura 3.9: Um exemplo de imagem do conjunto a ser classificado

Agora que entendemos o conjunto com o qual estamos trabalhando, vamos codificar as partes de *machine learning* do nosso algoritmo.

Primeiro, vamos inicializar algumas variáveis para armazenarmos pequenas cargas - fatias que compõe o conjunto completo - dos dados de treinamento para quando estabelecermos nosso modelo. A razão para trabalhar com pequenas cargas é para facilitar para o armazenamento da memória, ao invés de armazenar todo o conjunto. E essa ideia de trabalhar com cargas menores (aleatórias) de entrada, ao invés de todo o conjunto, nos leva para o passo do *Gradiente Descendente Estocástico*.

```
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])
```

Definimos nosso modelo linear para a função de valor após inserirmos dois parâmetros, **W** e **b**, representado no modelo da Figura 3.10.

```
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

Definimos nossa função de perda, para medir o quão bem ou mal foi o desempenho do modelo nas imagens com rótulos conhecidos, usando a chamada *perda de entropia cruzada*.

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_*tf.log(y),
    reduction_indices=[1]))
```

Usando os algoritmos de otimização que já vem no *TensorFlow*, podemos definir o otimizador gradiente descendente estocástico (para melhorar nossos parâmetros da nossa função de valor e minimizar a perda) em uma linha de código.

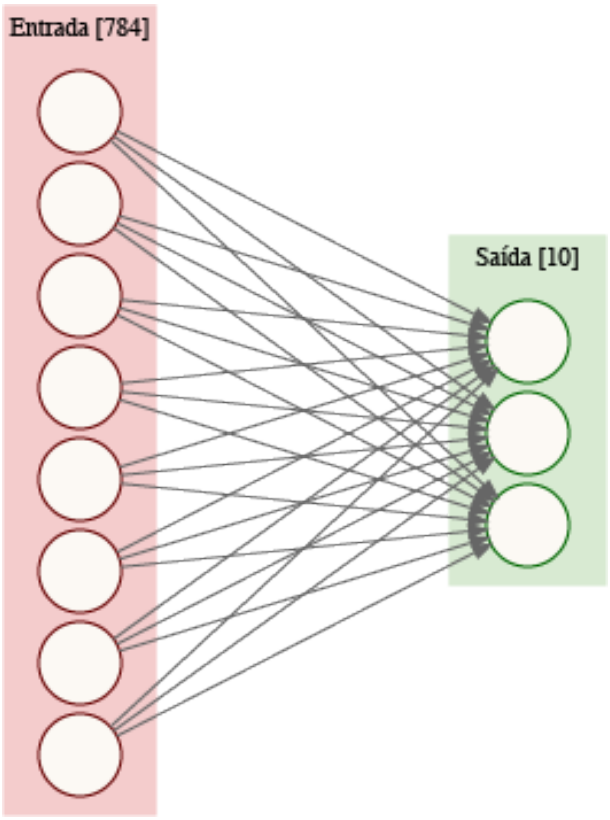


Figura 3.10: Esquema do modelo linear

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(
    cross_entropy)
```

Pela forma como o *TensorFlow* funciona, nós não executamos de fato nenhum código até agora. Tudo o que fizemos foi definir o que será chamado pelo algoritmo. Agora podemos começar a sessão para treinar o modelo e avaliar seu desempenho.

```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
```

Nós treinamos o modelo iterando por 1.000 passos, com uma carga de imagens definida aleatoriamente a cada passo.

```
for i in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

Agora nosso modelo está treinado. Podemos determinar sua acurácia passando como entrada as imagens de teste e os rótulos, buscando achar nossos próprios rótulos, e medindo os resultados. O código

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test
    .labels}))
```

retorna

```
0.9205
```

Cerca de 92% de acerto. Muito bom, não?

3.2.2 Agora com redes neurais

O classificador linear de imagens que nós acabamos de construir funciona surpreendentemente bem para o conjunto de dados MNIST (acurácia de 92%). Isso *parece muito bom*, pois isso implica que essa imagens são (em sua maioria) linearmente separáveis no espaço dimensional 784 no qual elas estão representadas. Isso significa que você pode traçar 10 linhas ($n-1$ hiperplanos dimensionais, na verdade) em um plano (um espaço n-dimensional, na verdade) com esses algarismos e separá-los facilmente em categorias, como na Figura 3.11.

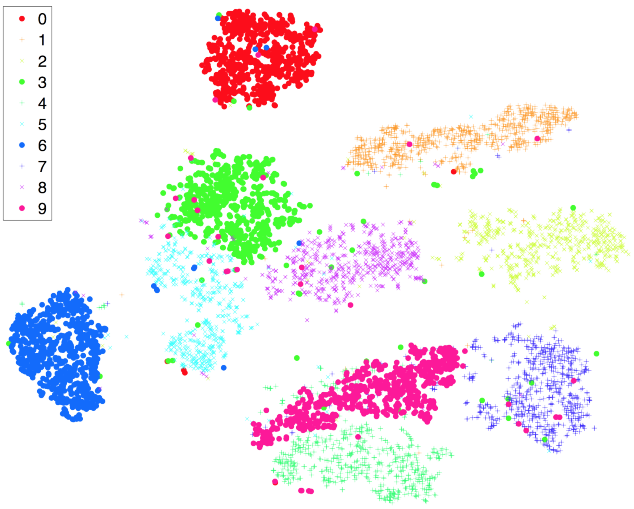


Figura 3.11: Um gráfico t-SNE do conjunto MNIST aproximando a representação dos algarismos em um plano 2D preservando a topologia dos dados [32]

Mas se você ler no tutorial do *TensorFlow* que lida com esse mesmo conjunto [30], eles afirmam categoricamente: "Obter acurácia de 92% no MNIST é ruim. É quase que vergonhosamente ruim."

Para não nos envergonharmos mais e melhorarmos o desempenho do nosso classificador, precisamos de uma função de valor mais genérica e não-linear. A boa notícia é que, ao generalizarmos a função de valor, o resto do que fizemos (a função de perda e o processo de otimização) permanece o mesmo!

Para fazer a função de valor ser não-linear, precisamos introduzir o conceito de *neurônio*. Ele está representado pela Figura 3.12, e faz três coisas:

1. Multiplica cada entrada por um peso;
2. Soma essas entradas e adiciona uma tendência;

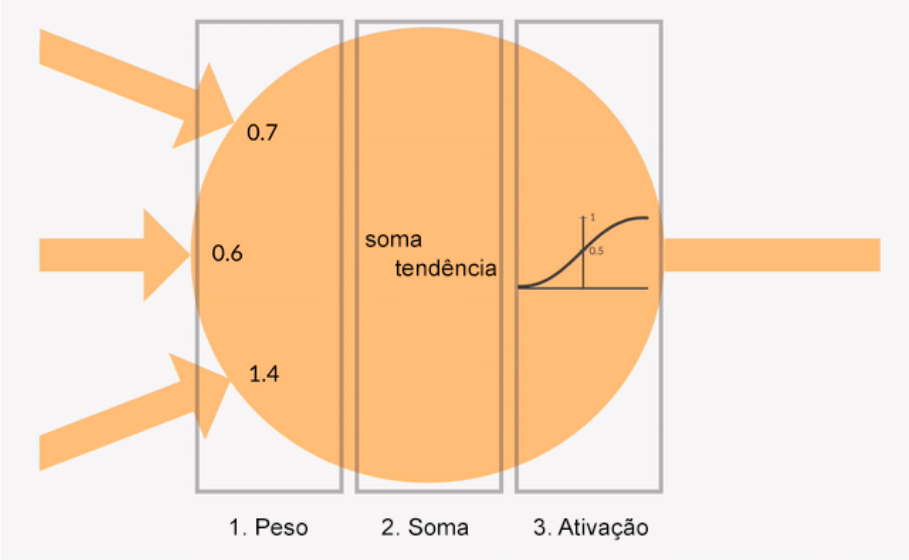


Figura 3.12: Modelo de neurônio artificial [33]

3. Passa esse valor por uma função não-linear chamada de *ativação* e produz uma saída.

Esses neurônios podem ser organizados em camadas, para formar uma *rede neural*, que nas suas camadas mais externas tem a mesma forma da nossa entrada e da nossa saída. Para o conjunto MNIST, com o qual estamos trabalhando, isso significa um vetor de 784 números entrando e 10 números saindo. A camada do meio é chamada de *camada oculta*, uma vez que não a acessamos diretamente pela entrada ou pela saída. Ela pode ser de um tamanho arbitrário, e isso é o tipo de coisa que define a *arquitetura da rede*. No jargão das redes neurais, a rede representada pela Figura 3.13 é chamada de *rede de duas camadas* ou *rede de uma camada oculta* - e nós podemos usar quantas camadas ocultas acharmos necessário.

Ao empilhar os neurônios dessa forma, podemos expressar de forma direta as operações realizadas pela rede em uma notação vetor-matriz:

1. Recebe um vetor $\mathbf{x}$ de entrada;
2. A camada oculta primeiro realiza uma operação linear com alguns pesos e tendências,

$$\mathbf{y}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1,$$

seguida da não-linearidade mais simples que se pode imaginar, uma função linear chamada *ativação linear retificada* (ReLU):

$$\mathbf{h}_1 = \max(0, \mathbf{y}_1).$$

(Existem outras formas funcionais que se poderia usar para essa função de ativação não-linear, mas essa é realmente popular atualmente e basta para a nossa necessidade).

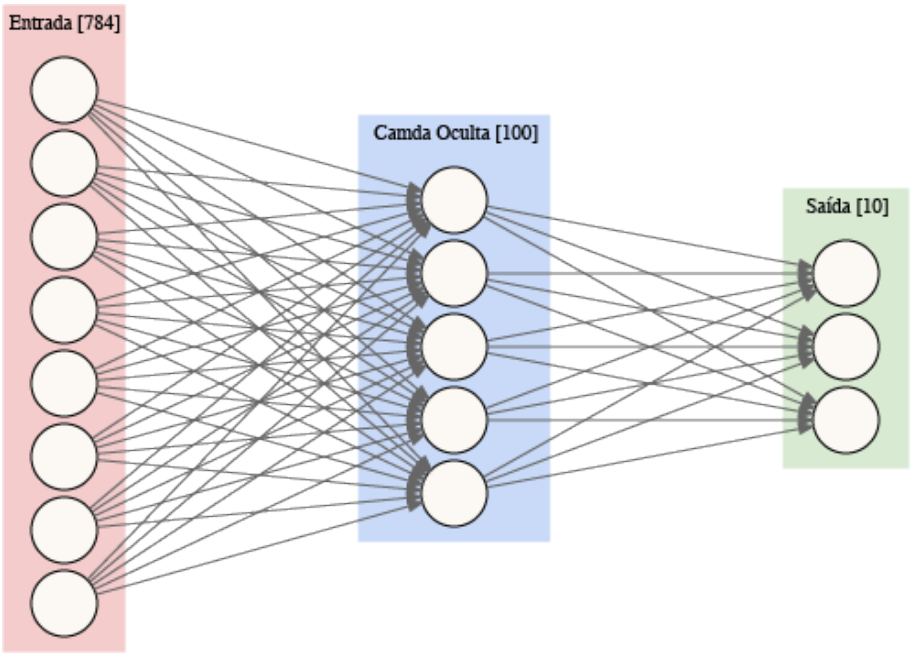


Figura 3.13: Empilhando neurônios em uma rede neural com uma camada oculta.

3. Finalmente, empilhamos uma operação matriz-vetor linear (introduzindo um outro conjunto de pesos e tendências) para trazer nossa saída de volta para o tamanho que queremos, 10 números.

$$\mathbf{y}_2 = \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2$$

Assim como antes, passamos essa saída por uma função *softmax* σ para que possamos interpretar a saída como a probabilidade de cada categoria.

$$\mathbf{s} = \sigma(\mathbf{y}_2)$$

Isso é o mais básico de redes neurais. Se você olhar com atenção, verá que elas são apenas conjuntos de multiplicações de matrizes misturadas com funções de ativação não-lineares.

Exercício 2: Um classificador baseado em redes neurais para o MNIST

Agora vamos estender nosso classificador de imagens para um baseado em redes neurais, e esperamos ver ganhos consideráveis na acurácia da classificação nesse processo.

Lembre-se que, no caso linear que vimos anteriormente, o modelo para a função de valor era:

```
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

Agora vamos redefinir essa função como um modelo não-linear com apenas algumas linhas a mais de código. O nosso modelo é uma rede neural de uma camada oculta, como vimos anteriormente. Isso introduz dois conjuntos de parâmetros, **W1**, **b1**, e **W2**, **b2**:

```
W1 = tf.Variable(tf.zeros([784, 100]))
b1 = tf.Variable(tf.zeros([100]))
h1 = tf.nn.relu(tf.matmul(x, W1) + b1)

W2 = tf.Variable(tf.zeros([100, 10]))
b2 = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(h1, W2) + b2)
```

Como antes, nós agora definimos a função de perda de entropia cruzada, que quantifica o desempenho desse modelo em imagens com rótulos conhecidos e usamos o otimizador gradiente descendente estocástico para melhorar iterativamente os parâmetros e minimizar a perda. Uma vez que esse modelo é treinado, podemos passar imagens de teste e rótulo para ele e determinar a acurácia média.

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test
    .labels}))
```

```
0.1135
```

11%?! Nós esperávamos obter enormes ganhos sobre os 92% do modelo anterior, mas o que recebemos foi algo terrível. *Por quê?* Por que, depois de toda essa conversa de aproximações e ativações e redes neurais, nosso resultado é tão ruim?

A verdade é que, *só porque a nossa função de valor pode, teoricamente, calcular qualquer coisa, isso não significa que o nosso algoritmo de aprendizado sempre vai conseguir calcular.* E se você olhar para a definição do modelo nos códigos acima, você perceberá que isso acontece porque nós inicializamos todos os nossos pesos e tendências como 0. E, por causa disso, pela forma com que as ReLUs são definidas, iniciar tudo com 0 significa que elas estão *mortas* (não são executadas), e assim o gradiente descendente não tem como melhorar os parâmetros.

Isso explica o resultado de acurácia de 11%, que, se você pensar bem, é basicamente o que você conseguiria se tentasse adivinhar aleatoriamente entre 10 categorias. Então, o que nós precisamos é melhorar a inicialização desses parâmetros:

```
W1 = tf.Variable(tf.truncated_normal(shape=[784, 100], stddev=0.1))
b1 = tf.Variable(tf.constant(0.1, shape=[100]))
h1 = tf.nn.relu(tf.matmul(x, W1) + b1)

W2 = tf.Variable(tf.truncated_normal(shape=[100, 10], stddev=0.1))
b2 = tf.Variable(tf.constant(0.1, shape=[10]))
y = tf.nn.softmax(tf.matmul(h1, W2) + b2)
```

Inicializamos os pesos com valores *normais* (literalmente, escolhemos valores de uma distribuição *normal*, truncados entre dois desvios padrões ao redor da média) e inicializamos as tendências com uma pequena constante positiva, 0.1.

Com essa mudança aparentemente trivial, retreinar o modelo e verificar sua acurácia média nos dados de teste contam uma história muito diferente:

```
0.9654
```

Mais de 96%! Muito melhor.

3.2.3 Finalmente, as redes neurais convolucionais

Estamos agora muito bem preparados, em termos de entendimento e habilidades. Conseguimos construir uma rede neural (de duas camadas) que faz um ótimo trabalho de classificação de imagens (mais de 96% de acerto nos dados do MNIST). E talvez você tenha percebido que as últimas melhorias não foram mais do que algumas linhas de código a mais do que o classificador linear que construímos inicialmente.

Se eu te perguntasse como poderíamos melhorar ainda mais a acurácia do nosso classificador, você talvez chutasse que poderíamos facilmente aumentar o número de camadas da nossa função de valor (isso é, fazer o nosso modelo mais *profundo*). Talvez algo como a Figura 3.14.

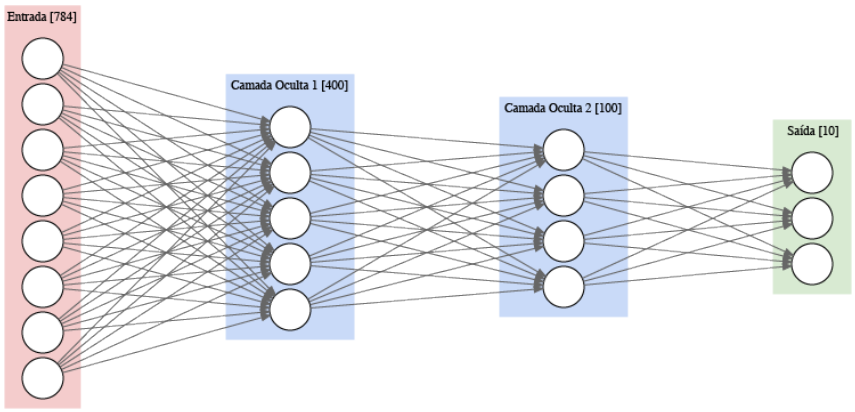


Figura 3.14: Uma rede neural de duas camadas ocultas.

A razão por trás desse chute é que, mesmo que cada neurônio seja apenas minimamente não-linear, introduzir mais camadas deles permite mais não-linearidade, aumentando a capacidade de aproximação da função de valor. Embora isso seja, de fato, verdade, há alguns obstáculos fundamentais com a forma geral das redes neurais que vimos até agora (o termo para elas é *redes neurais padrão* ou *integralmente conectadas*):

Primeiro, elas desconsideram completamente a estrutura 2D da imagem. Lembre que, ao invés de trabalhar com a entrada como uma matriz de 28x28, elas trabalhavam com a entrada de um vetor de 784 números. E você pode imaginar que há informações úteis nos *pixels* mais próximos uns dos outros que está sendo perdida.

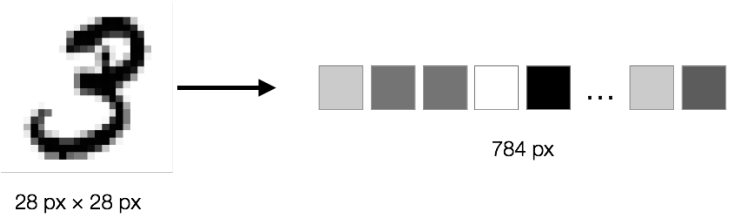


Figura 3.15: Redes neurais padrão ignoram a estrutura da imagem.

Em segundo lugar, o número de parâmetros que precisaríamos aprender cresce rapidamente conforme adicionamos mais camadas. Aqui estão o número de parâmetros correspondentes aos exemplos que vimos até agora:

- **Linear**: 784*10 + 10 = 7,850
- **Rede neural (uma camada oculta)**: 784*100 + 100 + 100*10 + 10 = 79,510
- **Rede neural (duas camadas ocultas)**: 784*400 + 400 + 400*100 + 100 + 100*10 + 10 = 355,110

A razão básica para esse rápido crescimento nos parâmetros é que cada neurônio em uma dada camada vê todos os neurônios da camada anterior. Além disso, você pode imaginar o quão pior isso seria se a nossa imagem não fosse minúscula (28px × 28px), mas tivesse um tamanho mais próximo da realidade.

Redes neurais convolucionais (*convnets*) foram projetadas para resolver esses dois problemas, tornando-as particularmente poderosas para lidar com dados de imagens. E, como veremos em breve, podemos usá-las para construir um classificador de imagens *profundo* estado da arte.

Arquitetura das redes convolutivas

As *convnets* são muito similares às redes neurais integralmente conectadas, em que elas são feitas de neurônios organizados em camadas e eles possuem pesos e tendências otimizáveis. Cada neurônio recebe algumas entradas, faz um mapeamento linear e aplica opcionalmente uma ativação não linear. Toda a rede, em ambos os casos, expressa uma única função de valor: dos *pixels* da imagem em uma ponta aos valores de categoria da outra.

O que torna as *convnets* especiais é que elas assumem explicitamente algumas informações sobre a estrutura da entrada - imagens, no nosso caso - que permite que elas possam representar certas propriedades em sua arquitetura. Essas escolhas as tornam eficientes em sua implementação e reduzem amplamente a quantidade de parâmetros na rede, como veremos em breve.

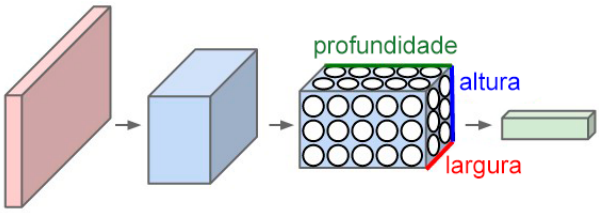


Figura 3.16: Arquitetura básica de uma Rede Neural Convolutiva [22]

Ao invés de lidar com os dados de entrada (e organizar camadas intermediárias de neurônios) como vetores lineares, elas tratam essas informações como volumes tridimensionais (com largura, altura e profundidade), isso é, cada camada recebe como entrada um volume 3D de números e produz como saída um volume 3D de números. O que seria uma imagem de entrada 2D ($L \times A$), se torna 3D ao introduzir a profundidade de cores como a terceira dimensão ($L \times A \times p$). Lembre que esse p seria 1 para os dados do MNIST, por eles serem em escala de cinza, mas para uma imagem colorida em RGB, a profundidade seria 3. De forma similar, a saída linear de largura C passa a ser presentada por $1 \times 1 \times C$.

Para fazer isso, as *convnets* se utilizam de dois tipos diferentes de camadas.

Camadas convolucionais A primeira é a *camada convolucional*, e você pode pensar nela como um conjunto de filtros otimizáveis. Se tivermos **K** filtros, cada filtro é um pequeno espaço, de extensão **F** e profundidade igual à profundidade da entrada. Isso é, um filtro típico seria $3 \times 3 \times 3$ ($\mathbf{F} = 3$ *pixels* de largura e altura, e 3 de profundidade por causa da entrada em 3 canais da imagem.

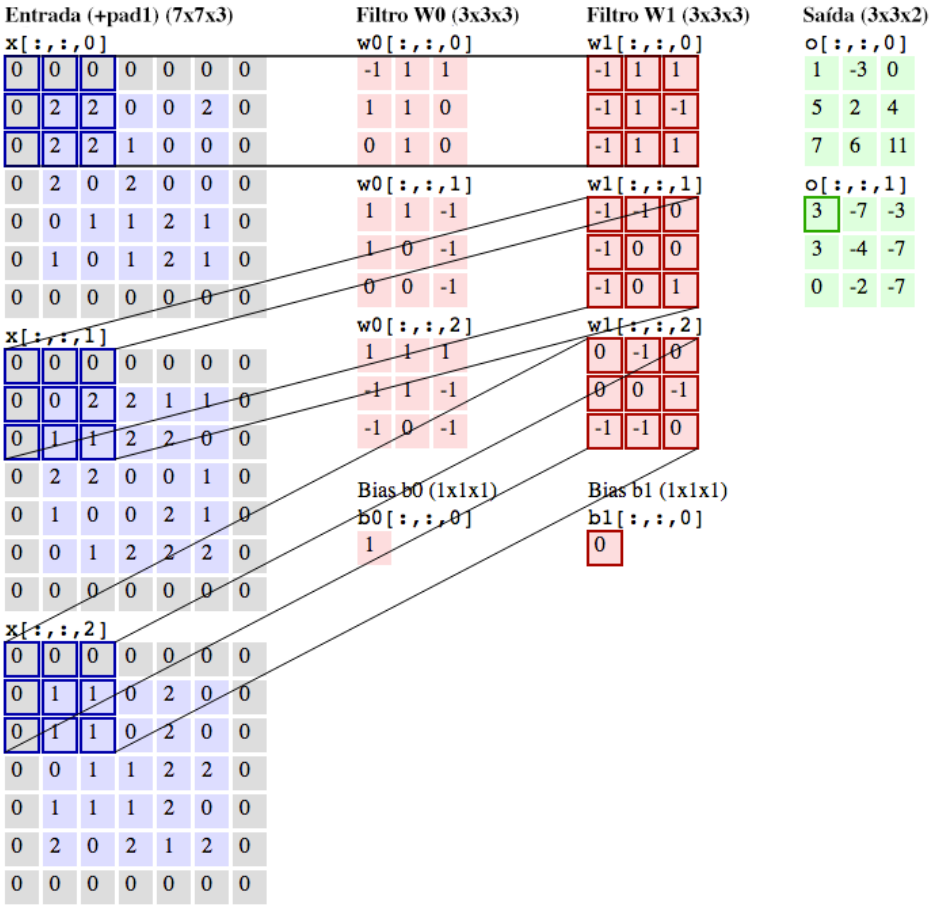


Figura 3.17: Representação de uma camada convolutiva com K = 2 filtros, cada um com extensão espacial F = 3 [22]

Seguindo a Figura 3.17, o filtro "desliza", ou *convolve* sobre o volume de entrada. Essa entrada pode ter uma borda espacial de zeros para controlar o espaço que o filtro irá percorrer. Ao avançar, cada filtro computa algum tipo de produto escalar com a entrada para produzir uma saída 2D, e quando empilhamos esses produtos ao longo dos filtros, temos um volume de saída 3D. Isso parece um pouco complicado à primeira vista, mas é uma ideia bastante simples e você vai se acostumar com o tempo.

O mais interessante para se notar aqui é:

1. Como esse conjunto de filtros é o mesmo conforme variamos a posição espacial, uma vez que eles aprendem a perceber uma certa característica, uma linha inclinada, por exemplo, eles aprendem a reconhecer essa característica em qualquer posição. Isso é, características espalhadas ao redor da imagem de entrada são tratadas da mesma forma.

2. Nós já não estamos mais aumentando o número de parâmetros loucamente, mesmo que o tamanho da imagem de entrada aumente muito ou o número de camadas aumente. Tudo que precisamos aprender são os pesos e as tendências correspondentes ao nosso conjunto de filtros (em vermelho na Figura 3.17), que são particularmente pequenos, por causa de seu tamanho espacial reduzido.

Camadas de agrupamento O segundo tipo importante de camada nas *convnets* é a *camada de agrupamento*. Essa é muito mais fácil de entender, pois ela se comporta apenas como um filtro de redução de amostragem. Essas camadas são usadas para reduzir o custo computacional do modelo e, até certo ponto, também reduzir algum sobreajuste. Note que ela não tem nenhum tipo de parâmetro para ser otimizado.

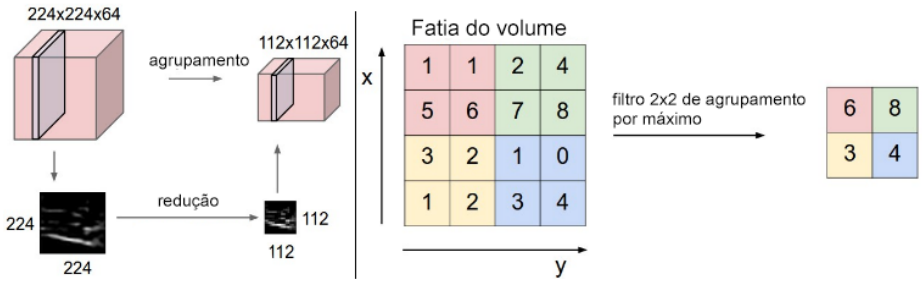


Figura 3.18: Exemplo da arquitetura de uma camada de agrupamento [22]

No exemplo da Figura 3.18, uma camada de agrupamento por máximo com extensão espacial $F = 2$ diminui pela metade a dimensão espacial da entrada, de 4×4 para 2×2, mantendo a profundidade a mesma. Ela faz isso ao escolher o valor máximo para cada conjunto 2×2 e passando esses para a saída. Você só precisa de uma camada de agrupamento para cada fatia de profundidade da entrada para cobrir todo o volume de entrada. Também é possível usar outros tipos de agrupamento, como *agrupamento por média*.

Exercício 3: Um classificador em *convnets* para o MNIST

Agora que vimos a base teórica dos dois tipos de camadas que formam a base das *convnets*, convolucional e agrupamento, teremos uma ideia melhor ao usá-las em um exemplo prático. Vamos voltar a estender o nosso classificador de algarismos MNIST com essas duas camadas adicionais e ver que tipo de ganho obtemos na acurácia da classificação.

Antes de irmos para o código, vamos reintroduzir algumas notações abstratas para explicar melhor as mudanças que estamos fazendo:

```
CC = Camada conectada
Softmax = Camada Softmax
ReLU = Unidade de Ativação Linear Retificada
Conv = Camada Convolucional
Grup = Camada de Agrupamento
```

Usando essa notação, os modelos que usamos até agora para a função de valor são:

1. **Linear:** CC → Softmax
2. **Rede neural (uma camada oculta):** CC → ReLU → CC → Softmax

O modelo baseado em redes convolucionais que vamos montar adiante é:

Conv → ReLU → Grup → CC → ReLU → CC → Softmax

```
# Definicao de algumas funcoes de utilidade para facilitar a montagem do
    modelo
def weight_variable(shape):
    return tf.Variable(tf.truncated_normal(shape, stddev=0.1))

def bias_variable(shape):
    return tf.Variable(tf.constant(0.1, shape=shape))

def conv(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def pool(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
   padding='SAME')

# A funcao de valor (o modelo) e' composta de algumas camadas

# Modela a entrada para virar um volume 3D
x_image = tf.reshape(x, [-1, 28, 28, 1])

# Uma camada convolucional (CONV -> RELU -> GRUP)
W_conv = weight_variable([5, 5, 1, 32])
b_conv = bias_variable([32])
h_conv = tf.nn.relu(conv(x_image, W_conv) + b_conv)
h_pool = pool(h_conv)

# Uma camada integralmente conectada (CC)
W_fc1 = weight_variable([14*14*32, 1024])
b_fc1 = bias_variable([1024])
h_pool_flat = tf.reshape(h_pool, [-1, 14*14*32])
h_fc1 = tf.nn.relu(tf.matmul(h_pool_flat, W_fc1) + b_fc1)

# Outra camada conectada (para "leitura") (CC)
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y = tf.nn.softmax(tf.matmul(h_fc1, W_fc2) + b_fc2)
```

Como antes, definimos a função de perda de entropia cruzada, que quantifica o desempenho desse modelo em imagens com rótulos conhecidos e usamos um otimizador para melhorar os parâmetros iterativamente e minimizar a perda. Uma vez que esse modelo é treinado, podemos passar as imagens de teste e os rótulos e medir sua acurácia média.

```
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test
    .labels}))
```

```
0.989
```

Quase 99% de acerto! Excelente!

Como as *convnets* funcionam (tão bem)?

Se você teve um pouco de dificuldade para entender o que estava acontecendo no Exercício 3, para entender como o classificador baseado em *convnets* funciona, a página da biblioteca *Keras.js*[6] disponibiliza um exemplo interativo desse mesmo classificador e os resultados de cada etapa do processo.

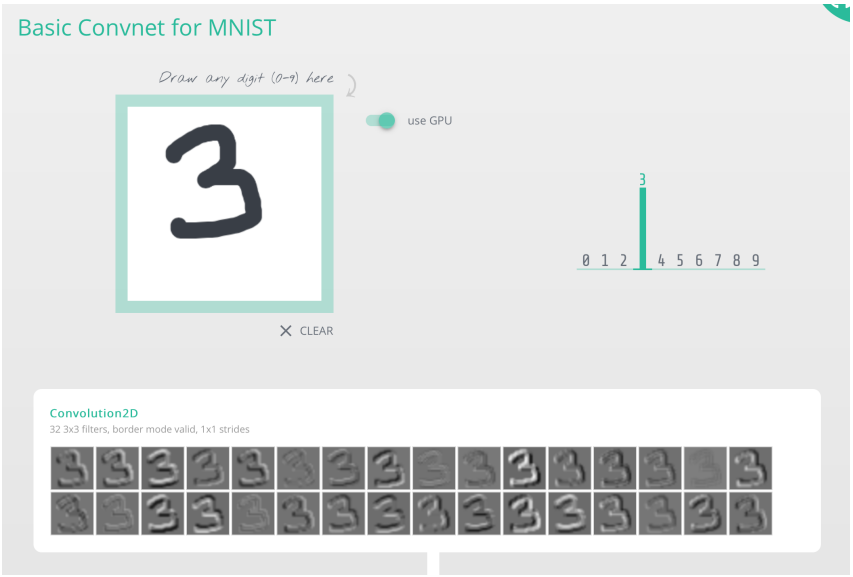


Figura 3.19: Classificador interativo de algarismos MNIST

Você vai brincando e desenhando qualquer algarismo que quiser. Ao fazer, a página te ajuda a visualizar as saídas das diferentes camadas, levando até a classificação (*softmax*) no final. Você começa a perceber, ao avançar nas camadas, que o sistema basicamente transforma as representações da entrada em formas que fazem mais sentido para a tarefa de classificação.

Para reforçar esse ponto, veja um outro exemplo de classificador *convnet* na Figura 3.20

Dada uma entrada de *pixels*, a primeira camada oculta é estimulada por características simples como bordas, a próxima talvez por coisas como contornos, e a próxima talvez por formas mais simples e partes de objetos. Quanto mais fundo você vai, mais elas começam a entender todo o campo da entrada, não apenas uma região pequena, mas, mais importante ainda, mais perto elas se movem em direção a *uma representação que torna mais fácil para classificar*.

Parece que as redes neurais convolucionais são, basicamente, *aprender representações*. E é isso que as torna tão poderosas.

Na verdade, isso é verdade para todo o campo do *deep learning* em geral (nós chamamos esses modelos de *deep*, "profundos", conforme vamos empilhando mais camadas). Mas as *convnets* são usadas massivamente como exemplos de ensino (como estamos fazendo agora) porque cada camada faz algo, no mínimo, reconhecível, para nós humanos.

[6] https://transcranial.github.io/keras-js/#/mnist-cnn

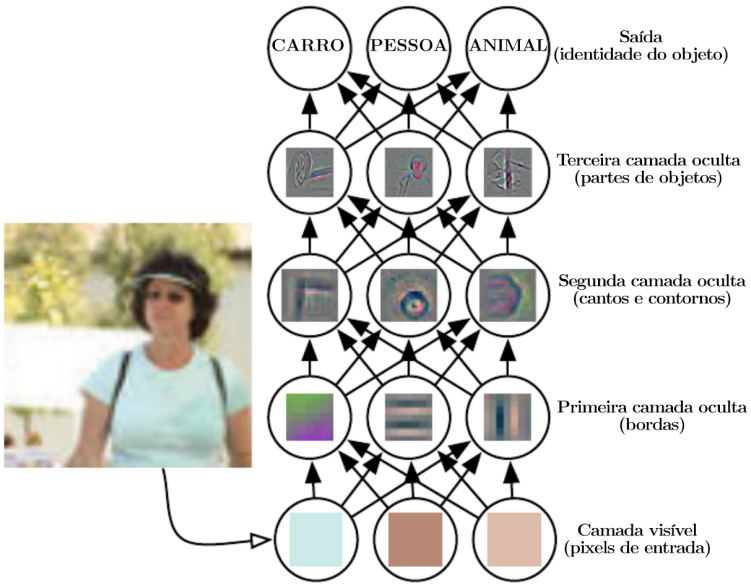


Figura 3.20: *Convnets* (e *deep learning* em geral) diz respeito basicamente a aprender representações [28]

3.3 Retornando ao problema de transferência de estilos

Agora que temos o vocabulário para falarmos sobre *convnets* e entendemos algo de como elas se comportam, *finalmente* podemos retornar ao problema original, da transferência de estilo, que motivou toda essa jornada.

Lembre, nós tínhamos uma imagem de conteúdo **c**, uma imagem de estilo **s**, e o objetivo é, de alguma forma, extrair o estilo (as cores, as texturas, etc) de **s** e aplicar ao conteúdo (a estrutura) de **c**, para chegar em um resultado semelhante ao da Figura 3.21.

Figura 3.21: À esquerda, a imagem **c** de onde vem o conteúdo, a imagem **s**, de onde vem o estilo e, à direita, a imagem **x**, com o estilo transferido, gerada pelo Prisma

Lembre também que, conforme o artigo de Gatys *et al.*, podemos expressar esse problema como um problema de otimização:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \left(\alpha \mathcal{L}_{\text{conteudo}}(\mathbf{c}, \mathbf{x}) + \beta \mathcal{L}_{\text{estilo}}(\mathbf{s}, \mathbf{x})\right)$$

Isto é, estamos procurando uma imagem $\mathbf{x}$ que difere o mínimo possível *em termos de conteúdo* da imagem de conteúdo $\mathbf{c}$ enquanto, simultaneamente, difere o mínimo possível, *em termos de estilo*, da imagem de estilo $\mathbf{s}$. O único problema nesse plano é que nós não tínhamos uma forma conveniente de medir as diferenças entre imagens em termos de conteúdo e estilo.

Mas agora nós temos!

Pelo que vimos, as *convnets* pré-treinadas para classificação de imagens *já aprenderam a codificar as informações perceptivas e semânticas que precisamos para medir a diferença entre esses termos*! A explicação principal para isso é que, ao aprender a reconhecer objetos, *a rede se torna invariante para qualquer variação de imagem que é supérflua para a identidade do objeto.*

Assim, daqui em diante, nós vamos nos familiarizar com um tipo particular de classificador de imagens baseado em *convnets* profundas chamado VGGNet [34], e então veremos como podemos usá-lo para o problema de transferência de estilo.

3.3.1 O classificador VGGNet

Quando finalizamos o exercício 3, estávamos utilizando um classificador de imagens baseado em *convnets* simples, que fez um ótimo trabalho com o conjunto MNIST. Agora vamos ver uma família de arquiteturas mais poderosa, chamada VGGNet, que é capaz de trabalhar com dados de forma bem genérica: imagens coloridas de objetos variados. Essa rede é a base do artigo de Gatys *et al.* sobre transferência de estilo, como veremos em breve.

Talvez você tenha reparado que, quando implementamos a nossa *convnet*, por mais simples que ela fosse, era difícil escrever (em termos de código reutilizável, e de acompanhar as formas dos objetos que eram passados entre os módulos do código) e lenta para treinar. Para evitarmos isso, não vamos escrever uma rede VGGNet do nada. Ao invés disso, vamos usar uma versão que alguém já escreveu e já pré-treinou.

Para simplificar nossa vida ainda mais, não vamos mais usar a biblioteca *TensorFlow*, propriamente dita, daqui para frente - vamos usar uma biblioteca de nível mais alta chamada *Keras*. Veremos tudo isso em prática no exercício a seguir.

Exercício 4: Utilizando e aplicando uma rede VGGNet (16) no Keras

Vamos começar importando alguns pacotes necessários. Note que, mesmo pretendendo utilizar o *Keras*, ele usa o *TensorFlow* para operações de baixo nível como produtos *tensor* e convoluções.

```
import numpy as np
from PIL import Image

from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input,
    decode_predictions
```

Nesse exercício, vamos buscar uma rede que já foi treinada com o conjunto ImageNet [35]. Em particular, vamos usar o modelo VGGNet de 16 camadas (**VGG16**, daqui em diante).

O projeto ImageNet é um vasto banco de dados visual projetado para ser usado em pesquisas de *softwares* de reconhecimento de objetos. Atualmente, conta com mais de 10 milhões de URLs de imagens que foram rotuladas para identificar os objetos representados por elas - um processo colaborativo supervisionado pelo ImageNet.

Figura 3.22: Uma amostra do conjunto ImageNet

Começando em 2010, o Desafio ImageNet de Reconhecimento Visual em Larga Escala é uma competição onde times de pesquisa submetem aplicações que classificam e detectam objetos e cenas do conjunto ImageNet (seriam os Jogos Olímpicos da visão computacional!).

A VGGNet foi introduzida como um dos concorrentes no Desafio de 2014 e garantiu o primeiro e o segundo lugar nas disputas de localização e classificação, respectivamente. Posteriormente, foi descrita em detalhes em um artigo publicado no ano seguinte [34].

Esse artigo descreve como uma família de modelos basicamente composta de filtros convolucionais simples (3×3) com profundidade crescente (de 11 a 19 camadas) conseguiu um desempenho tão bom em uma gama de tarefas de visão computacional.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figura 3.23: A arquitetura de redes VGGNet, destacando a variante de 16 camadas

A primeira coisa que vamos fazer é reproduzir a variante de 16 camadas (conforme destaque na Figura 3.23) e, no próximo exercício, vamos usá-la para o nosso problema de transferência de estilo.

Fazer isso é trivial no *Keras*, em apenas uma linha. Há uma vasta seleção de modelos possíveis de serem importados.

```
model = VGG16(weights='imagenet', include_top=True)
```

Vamos olhar esse modelo, só para garantir que é o mesmo descrito no artigo. O comando

```
layers = dict([(layer.name, layer.output) for layer in model.layers])
layers
```

retorna

```
{'block1_conv1': <tf.Tensor 'Relu:0' shape=(?, 224, 224, 64) dtype=
    float32>,
```

```
 'block1_conv2': <tf.Tensor 'Relu_1:0' shape=(?, 224, 224, 64) dtype=
   float32>,
 'block1_pool': <tf.Tensor 'MaxPool:0' shape=(?, 112, 112, 64) dtype=
   float32>,
 'block2_conv1': <tf.Tensor 'Relu_2:0' shape=(?, 112, 112, 128) dtype=
   float32>,
 'block2_conv2': <tf.Tensor 'Relu_3:0' shape=(?, 112, 112, 128) dtype=
   float32>,
 'block2_pool': <tf.Tensor 'MaxPool_1:0' shape=(?, 56, 56, 128) dtype=
   float32>,
 'block3_conv1': <tf.Tensor 'Relu_4:0' shape=(?, 56, 56, 256) dtype=
   float32>,
 'block3_conv2': <tf.Tensor 'Relu_5:0' shape=(?, 56, 56, 256) dtype=
   float32>,
 'block3_conv3': <tf.Tensor 'Relu_6:0' shape=(?, 56, 56, 256) dtype=
   float32>,
 'block3_pool': <tf.Tensor 'MaxPool_2:0' shape=(?, 28, 28, 256) dtype=
   float32>,
 'block4_conv1': <tf.Tensor 'Relu_7:0' shape=(?, 28, 28, 512) dtype=
   float32>,
 'block4_conv2': <tf.Tensor 'Relu_8:0' shape=(?, 28, 28, 512) dtype=
   float32>,
 'block4_conv3': <tf.Tensor 'Relu_9:0' shape=(?, 28, 28, 512) dtype=
   float32>,
 'block4_pool': <tf.Tensor 'MaxPool_3:0' shape=(?, 14, 14, 512) dtype=
   float32>,
 'block5_conv1': <tf.Tensor 'Relu_10:0' shape=(?, 14, 14, 512) dtype=
   float32>,
 'block5_conv2': <tf.Tensor 'Relu_11:0' shape=(?, 14, 14, 512) dtype=
   float32>,
 'block5_conv3': <tf.Tensor 'Relu_12:0' shape=(?, 14, 14, 512) dtype=
   float32>,
 'block5_pool': <tf.Tensor 'MaxPool_4:0' shape=(?, 7, 7, 512) dtype=
   float32>,
 'fc1': <tf.Tensor 'Relu_13:0' shape=(?, 4096) dtype=float32>,
 'fc2': <tf.Tensor 'Relu_14:0' shape=(?, 4096) dtype=float32>,
 'flatten': <tf.Tensor 'Reshape_13:0' shape=(?, ?) dtype=float32>,
 'input_1': <tf.Tensor 'input_1:0' shape=(?, 224, 224, 3) dtype=float32
   >,
 'predictions': <tf.Tensor 'Softmax:0' shape=(?, 1000) dtype=float32>}
```

Parece estar certo. Agora vamos ter uma noção de quantos parâmetros há nesse modelo:

```
model.count_params()
```

```
138357544
```

Bastante! Nós acabamos de descobrir que nos livramos de treinar um modelo com mais de 138 milhões de parâmetros! Uma vez que o modelo já foi treinado, podemos usá-lo para classificação.

```
image_path = 'images/elephant.jpg'
image = Image.open(image_path)
image = image.resize((224, 224))
image
```

Figura 3.24: Imagem de teste

```
# Converte a imagem em vetor
x = np.asarray(image, dtype='float32')
# Converte esse vetor em uma serie de vetores
x = np.expand_dims(x, axis=0)
# Pre-processa a entrada para bater com os dados de treino
x = preprocess_input(x)
```

O código a seguir classifica a imagem de teste e decodifica os resultados em uma lista de tuplas (classe, descrição, probabilidade). Existe uma lista desse tipo para cada amostra do conjunto, mas como só estamos enviando uma imagem de teste, só recebemos um conjunto como saída.

```
preds = model.predict(x)
print('Palpite:', decode_predictions(preds, top=3)[0])

('Palpite:', [(u'n02504458', u'African_elephant', 0.84805286), (u'
    n01871265', u'tusker', 0.10270286), (u'n02504013', u'Indian_elephant'
    , 0.049056333)])
```

3.3.2 O algoritmo de transferência de estilo artístico

No processo de aprender como classificar imagens, o modelo VGG16 que acabamos de acessar e aplicar (no Exercício 4), aprendemos muito mais. Ao se treinar para transformar uma entrada de *pixels* em valores de categoria, a rede aprendeu a codificar uma boa porção de informações perceptivas e semânticas sobre imagens. O algoritmo neural de estilo introduzido em Gatys *et al.* mexe nessas representações para, primeiro, definir os termos semânticos de perda ($\mathcal{L}_{\text{conteudo}}(\mathbf{c}, \mathbf{x})$ e $\mathcal{L}_{\text{estilo}}(\mathbf{c}, \mathbf{x})$), e então usar esses termos para expor o problema de otimização para a transferência de estilo.

Exercício 5: Uma implementação concreta do algoritmo de transferência de estilo artístico

Como já falamos no começo, desde que Gatys *et al.* foi publicado, o alvoroço causado pela técnica proposta levou à implementação de diversas aplicações de código-aberto. Um dos exemplos mais populares e genéricos foi publicado por Justin Johnson [36]. Nosso algoritmo foi baseado na própria biblioteca de exemplos do *Keras* [24].

Como sempre, nós começamos importando alguns pacotes. Note que não são tantos.

```
from __future__ import print_function

import time
```

```
from PIL import Image
import numpy as np

from keras import backend
from keras.models import Model
from keras.applications.vgg16 import VGG16

from scipy.optimize import fmin_l_bfgs_b
from scipy.misc import imsave
```

Nossa primeira tarefa é carregar as imagens de conteúdo[7] e estilo[8]. Note que a imagem de conteúdo que estamos usando (Figura 3.25) não é uma foto particularmente de alta qualidade, mas a saída que vamos obter ainda parece muito boa.

```
height = 512
width = 512

content_image_path = 'images/karl_ove_knausgaard.jpg'
content_image = Image.open(content_image_path)
content_image = content_image.resize((width, height))

style_image_path = 'images/styles/wave.jpg'
style_image = Image.open(style_image_path)
style_image = style_image.resize((width, height))

content_image
style_image
```

Figura 3.25: Karl Ove Knausgaard e "A Grande Onda de Kanagawa"

Agora convertemos essas imagens em uma forma própria para processamento numérico. Em particular, nós acrescentamo uma nova dimensão (além da clássica forma *altura* × *largura* × 3), para que possamos concatenar, mais à frente, as representações dessas duas imagens em uma estrutura de dados em comum.

[7] Karl Ove Knausgaard, romancista norueguês
[8] A Grande Onda de Kanagawa, famosa xilogravura do mestre japonês Hokusai

```
content_array = np.asarray(content_image, dtype='float32')
content_array = np.expand_dims(content_array, axis=0)
print(content_array.shape)

style_array = np.asarray(style_image, dtype='float32')
style_array = np.expand_dims(style_array, axis=0)
print(style_array.shape)

(1, 512, 512, 3)
(1, 512, 512, 3)
```

Antes de avançar, precisamos manipular os dados de entrada para refletir o que foi feito no artigo [34] que introduziu o modelo das redes VGG que vamos usar. Para isso, precisamos realizar duas transformações:

1. Subtrair um valor RGB médio de cada *pixel* (computado previamente a partir no conjunto ImageNet),
2. Mudar a ordem do vetor RGB multidimensional de RGB para BGR (a ordem usada no artigo).

```
content_array[:, :, :, 0] -= 103.939
content_array[:, :, :, 1] -= 116.779
content_array[:, :, :, 2] -= 123.68
content_array = content_array[:, :, :, ::-1]

style_array[:, :, :, 0] -= 103.939
style_array[:, :, :, 1] -= 116.779
style_array[:, :, :, 2] -= 123.68
style_array = style_array[:, :, :, ::-1]
```

Agora estamos prontos para usar esses vetores para definir variáveis no *Keras*. Também introduzimos uma variável de preenchimento para guardar a imagem de combinação que registra o conteúdo da imagem de conteúdo e incorpora o estilo da imagem de estilo.

```
content_image = backend.variable(content_array)
style_image = backend.variable(style_array)
combination_image = backend.placeholder((1, height, width, 3))
```

Finalmente, concatenamos todos esses dados em um único *tensor* que é próprio para processar o modelo VGG16 do *Keras*.

```
input_tensor = backend.concatenate([content_image,
                                    style_image,
                                    combination_image], axis=0)
```

A ideia-chave introduzida por Gatys *et al.* é que as redes neurais convolucionais pré-treinadas para classificação de imagens já sabem como codificar informações perceptivas e semânticas das imagens. Nós vamos seguir essa ideia e usar o espaço de características providas por esse modelo para trabalhar independentemente com conteúdo e estilo de imagens.

O artigo original usa o modelo VGG de 19 camadas de Simonyan e Zisserman [34], mas vamos seguir Johnson *et al.* [37] e usar o modelo de 16 camadas. Não há uma diferença qualitativa notável ao fazer essa escolha, e ganhamos um pouco em velocidade.

Além disso, como não estamos (mais) interessados no problema da classificação, não precisamos das camadas de redes integralmente conectadas ou do classificador *softmax*. Nós só precisamos da parte do modelo marcada em verde na Figura 3.23.

É trivial para nós acessar esse modelo truncado porque o *Keras* vem com um conjunto de modelos pré-treinados, incluindo o modelo **VGG16**. Note que, ao passar o código *include_ top=False* no código a seguir, nós não incluímos nenhuma das camadas integralmente conectadas.

```
model = VGG16(input_tensor=input_tensor, weights='imagenet',
              include_top=False)
```

Como fica claro na Figura 3.23, o modelo com o qual nós estamos trabalhamos tem várias camadas. O *Keras* tem sua própria nomenclatura para essas camadas. Vamos listar esses nomes para que possamos fazer referência a eles individualmente:

```
layers = dict([(layer.name, layer.output) for layer in model.layers])
layers

{'block1_conv1': <tf.Tensor 'Relu:0' shape=(3, 512, 512, 64) dtype=
   float32>,
 'block1_conv2': <tf.Tensor 'Relu_1:0' shape=(3, 512, 512, 64) dtype=
   float32>,
 'block1_pool': <tf.Tensor 'MaxPool:0' shape=(3, 256, 256, 64) dtype=
   float32>,
 'block2_conv1': <tf.Tensor 'Relu_2:0' shape=(3, 256, 256, 128) dtype=
   float32>,
 'block2_conv2': <tf.Tensor 'Relu_3:0' shape=(3, 256, 256, 128) dtype=
   float32>,
 'block2_pool': <tf.Tensor 'MaxPool_1:0' shape=(3, 128, 128, 128) dtype=
   float32>,
 'block3_conv1': <tf.Tensor 'Relu_4:0' shape=(3, 128, 128, 256) dtype=
   float32>,
 'block3_conv2': <tf.Tensor 'Relu_5:0' shape=(3, 128, 128, 256) dtype=
   float32>,
 'block3_conv3': <tf.Tensor 'Relu_6:0' shape=(3, 128, 128, 256) dtype=
   float32>,
 'block3_pool': <tf.Tensor 'MaxPool_2:0' shape=(3, 64, 64, 256) dtype=
   float32>,
 'block4_conv1': <tf.Tensor 'Relu_7:0' shape=(3, 64, 64, 512) dtype=
   float32>,
 'block4_conv2': <tf.Tensor 'Relu_8:0' shape=(3, 64, 64, 512) dtype=
   float32>,
 'block4_conv3': <tf.Tensor 'Relu_9:0' shape=(3, 64, 64, 512) dtype=
   float32>,
 'block4_pool': <tf.Tensor 'MaxPool_3:0' shape=(3, 32, 32, 512) dtype=
   float32>,
 'block5_conv1': <tf.Tensor 'Relu_10:0' shape=(3, 32, 32, 512) dtype=
   float32>,
 'block5_conv2': <tf.Tensor 'Relu_11:0' shape=(3, 32, 32, 512) dtype=
   float32>,
 'block5_conv3': <tf.Tensor 'Relu_12:0' shape=(3, 32, 32, 512) dtype=
   float32>,
 'block5_pool': <tf.Tensor 'MaxPool_4:0' shape=(3, 16, 16, 512) dtype=
   float32>,
 'input_1': <tf.Tensor 'concat:0' shape=(3, 512, 512, 3) dtype=float32>}
```

Se você olhar a lista acima, você verá que cobrimos todos os itens que queríamos da tabela da arquitetura do modelo **VGG16**. Note também que, como já alimentamos o *Keras* com um *tensor* de entrada concreto, os vários tensores do *TensorFlow* já tem formas (*shapes*) bem definidos.

A grande ideia que estamos tentando entender e aplicar é que o problema da transferência de estilo pode ser expresso como um problema de otimização, onde a função de perda que queremos minimizar pode ser decomposta em três partes distintas: a *perda de conteúdo*, a *perda de estilo* e a *perda de variação total*.

A importância relativa desses termos é determinada por um conjunto de pesos escalares. Eles são arbitrários, mas o conjunto a seguir foi escolhido após bastante experimentação para encontrar um conjunto que gera uma saída que é esteticamente agradável.

```
content_weight = 0.025
style_weight = 5.0
total_variation_weight = 1.0
```

Agora vamos usar os espaços de características fornecidos pelas camadas específicas do nosso modelo para definir essas três funções de perda. Começamos inicializar a perda total como 0 e incrementá-la aos poucos.

```
loss = backend.variable(0.)
```

Perda de conteúdo Para a perda de conteúdo, seguimos Johnson *et al.* e retiramos as características de conteúdo do `block2_conv2`, porque a escolha original de Gatys *et al.* (`block4_conv2`) perde muito dos detalhes estruturais. Pelo menos para rostos e retratos, consideramos mais esteticamente agradável reter a estrutura da imagem de conteúdo original.

A variação ao longo das camadas é mostrada em alguns exemplos na Figura 3.26 (substituindo a notação `reluX_Y` com a notação `blockX_convY` do *Keras*).

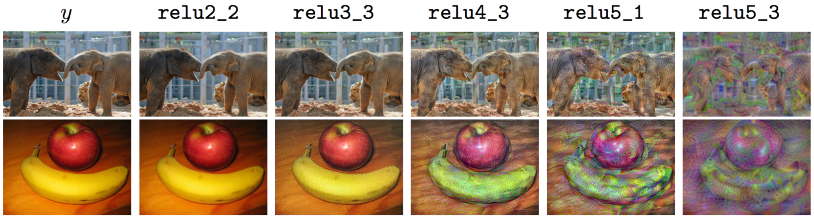


Figura 3.26: Reconstrução de características de conteúdo.

A perda de conteúdo é a distância Euclidiana (escalada e quadrada) entre as características de representação de conteúdo e as imagens combinadas.

```
def content_loss(content, combination):
    return backend.sum(backend.square(combination - content))

layer_features = layers['block2_conv2']
content_image_features = layer_features[0, :, :, :]
combination_features = layer_features[2, :, :, :]

loss += content_weight * content_loss(content_image_features,
```

```
                                        combination_features)
```

Perda de estilo Aqui a coisa começa a ficar complicada.

Para a perda de estilo, primeiro nós definimos algo chamado de *matriz Gram*. Os termos dessa matriz são proporcionais às covariâncias de conjuntos correspondentes de características, e assim capturam as informações cujas características tendem a se ativar juntas. Ao capturar apenas essas estatísticas agregadas ao longo da imagem, esses termos se tornam "cegos" ao arranjo dos objetos dentro da imagem. Isso é o que permite a eles capturar informações sobre estilo, independente de conteúdo. Isso não é de forma alguma trivial, e existem bons artigos que explicam essa ideia [38].

A matriz Gram pode ser computada eficientemente por meio de remodelar os espaços de características de forma apropriada e obter o produto diádico.

```
def gram_matrix(x):
    features = backend.batch_flatten(backend.permute_dimensions(x, (2,
   0, 1)))
    gram = backend.dot(features, backend.transpose(features))
    return gram
```

A perda de estilo então é a norma de Frobenius (escalada e quadrada) da diferença entre as matrizes Gram de estilo e da imagem combinada.

Novamente, no código a seguir, decidimos usar as características de estilo das camadas definidas em Johnson *et al.*, ao invés de Gatys *et al.*, por acharmos que os resultados finais são mais esteticamente agradáveis. Essas escolhas podem ser alteradas para se obter resultados diversos.

```
def style_loss(style, combination):
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = height * width
    return backend.sum(backend.square(S - C)) / (4. * (channels ** 2) *
   (size ** 2))

feature_layers = ['block1_conv2', 'block2_conv2',
                  'block3_conv3', 'block4_conv3',
                  'block5_conv3']
for layer_name in feature_layers:
    layer_features = layers[layer_name]
    style_features = layer_features[1, :, :, :]
    combination_features = layer_features[2, :, :, :]
    sl = style_loss(style_features, combination_features)
    loss += (style_weight / len(feature_layers)) * sl
```

Perda de variação total As coisas voltam a ficar mais simples agora.

Se você tentasse resolver o problema da otimização com apenas as duas perdas que já introduzimos (estilo e conteúdo), você obteria um resultado um tanto quanto ruidoso. Por isso, adicionamos mais um termo, chamado de perda de variação total, que leva a uma suavidade espacial.

Você pode experimentar reduzir o parâmetro `total_variation_weight` e alterar os níveis de ruído da imagem gerada.

```
def total_variation_loss(x):
    a = backend.square(x[:, :height-1, :width-1, :] - x[:, 1:, :width-1,
    :])
    b = backend.square(x[:, :height-1, :width-1, :] - x[:, :height-1,
   1:, :])
    return backend.sum(backend.pow(a + b, 1.25))

loss += total_variation_weight * total_variation_loss(combination_image)
```

O objetivo de todo esse trabalho era elaborar um problema de otimização cuja solução é uma *imagem combinada* que contém o conteúdo de uma imagem e o estilo de outra. Agora que já temos nossas imagens de entrada preparadas e nossos cálculos de função de perda no lugar, o que nos resta é definir gradientes para a perda total relativa à imagem combinada, e usar esses gradientes para otimizar iterativamente sobre nossa imagem combinada para minimizar a perda.

Nós começamos definindo nossos gradientes.

```
grads = backend.gradients(loss, combination_image)
```

Agora introduzimos uma classe `Evaluator`, que calcula perda e gradientes em uma passada, enquanto os recebe por meio de duas funções separadas, `loss` e `grads`. Isso é feito porque o `scipy.optimize` requer funções separadas para perda e gradientes, mas computá-los separadamente não seria eficiente.

```
outputs = [loss]
outputs += grads
f_outputs = backend.function([combination_image], outputs)

def eval_loss_and_grads(x):
    x = x.reshape((1, height, width, 3))
    outs = f_outputs([x])
    loss_value = outs[0]
    grad_values = outs[1].flatten().astype('float64')
    return loss_value, grad_values

class Evaluator(object):

    def __init__(self):
        self.loss_value = None
        self.grads_values = None

    def loss(self, x):
        assert self.loss_value is None
        loss_value, grad_values = eval_loss_and_grads(x)
        self.loss_value = loss_value
        self.grad_values = grad_values
        return self.loss_value

    def grads(self, x):
        assert self.loss_value is not None
        grad_values = np.copy(self.grad_values)
        self.loss_value = None
```

```
        self.grad_values = None
        return grad_values

evaluator = Evaluator()
```

Agora estamos finalmente prontos para resolver nosso problema de otimização. A imagem combinada começa sua vida como uma coleção aleatória de *pixels* (válidos), e usamos o algoritmo L-BFGS (um algoritmo *quasi*-Newtoniano que é significativamente mais rápido para convergir que o gradiente descendente padrão) para otimizar sobre ela.

Paramos após 10 iterações porque o resultado já é satisfatório a esse ponto e a perda deixa de ser significativa daí em diante. Note que nós precisamos passar nossa imagem de saída pelo inverso das transformações que fizemos com a entrada, para que ela possa fazer sentido visual. Cada iteração é salva em um arquivo cujo nome indica os parâmetros de perda de estilo, conteúdo e variação total, e o contador da iteração, para que possamos visualizar cada passo do processo separadamente.

```
if K.image_dim_ordering() == 'th':
    x = np.random.uniform(0, 255, (1, 3, img_nrows, img_ncols)) - 128.
else:
    x = np.random.uniform(0, 255, (1, img_nrows, img_ncols, 3)) - 128.

def deprocess_image(x):
    if K.image_dim_ordering() == 'th':
        x = x.reshape((3, img_nrows, img_ncols))
        x = x.transpose((1, 2, 0))
    else:
        x = x.reshape((img_nrows, img_ncols, 3))
    x = x[:, :, ::-1]
    x[:, :, 0] += 103.939
    x[:, :, 1] += 116.779
    x[:, :, 2] += 123.68
    x = np.clip(x, 0, 255).astype('uint8')
    return x

iterations = 10

for i in range(iterations):
    print('Inicio da iteracao', i)
    start_time = time.time()
    x, min_val, info = fmin_l_bfgs_b(evaluator.loss, x.flatten(),
                                     fprime=evaluator.grads, maxfun=20)
    print('Valor atual de perda:', min_val)

    img = deprocess_image(x.copy())
    fname = os.path.join('cw_%g_sw_%g_tvw_%g_i_%d.png' %
                         (content_weight,
                          style_weight,
                          total_variation_weight,
                          i))
    imsave(fname, img)
    end_time = time.time()
    print('Iteracao %d completa em %ds' % (i, end_time - start_time))
```

```
Inicio da iteracao 0
```

```
Valor atual de perda: 32336822000.0
Iteracao 0 completa em 143s
Inicio da iteracao 1
Valor atual de perda: 27063730000.0
Iteracao 1 completa em 142s
Inicio da iteracao 2
Valor atual de perda: 25835143000.0
Iteracao 2 completa em 142s
Inicio da iteracao 3
Valor atual de perda: 25137762000.0
Iteracao 3 completa em 142s
Inicio da iteracao 4
Valor atual de perda: 24676390000.0
Iteracao 4 completa em 142s
Inicio da iteracao 5
Valor atual de perda: 24361574000.0
Iteracao 5 completa em 143s
Inicio da iteracao 6
Valor atual de perda: 24131527000.0
Iteracao 6 completa em 142s
Inicio da iteracao 7
Valor atual de perda: 23957942000.0
Iteracao 7 completa em 142s
Inicio da iteracao 8
Valor atual de perda: 23821642000.0
Iteracao 8 completa em 143s
Inicio da iteracao 9
Valor atual de perda: 23712221000.0
Iteracao 9 completa em 143s
Inicio da iteracao 10
Valor atual de perda: 23621476000.0
```

Esse processo leva algum tempo, mesmo em um computador com GPU dedicada, então não se surpreenda se demorar bem mais em um sem.

A Figura 3.27 apresenta os resultados concatenados das 10 iterações, e o resultado final está apresentado na Figura 3.28.

Figura 3.27: A saída das dez iterações.

Figura 3.28: O resultado final.

3.4 Concluindo

Nossa jornada para a solução do problema de transferência de estilo chegou ao fim. Começamos em um lugar aparentemente desimportante, o problema da classificação de imagens. Em pouco tempo, percebemos que a distância semântica entre a representação das imagens (em *pixels*) e a tarefa que tentamos resolver tornava quase impossível escrever um classificador como um programa imperativo. Então nos voltamos para o aprendizado supervisionado, que é uma ótima ferramenta para quando você tem um problema intuitivo de expressar, mas que é difícil de elaborar uma lista de passos que o resolva.

Ao nos embrenharmos por soluções de aprendizado supervisionado cada vez mais sofisticadas para o problema da classificação de imagens, descobrimos que as redes neurais convolucionais são muito boas para isso. Não só isso, nós entendemos que elas tem um desempenho tão bom porque elas são boas em aprender representações dos dados de entrada próprios para a tarefa que queríamos cumprir. Elas fazem de forma praticamente automática, sem a necessidade de mexer explicitamente com características, o que, se fosse o caso, requer uma quantidade significativa de expertise nesses processos. Além disso, percebemos que o que nós realmente aprendemos, de certa forma, foi como enxergar! É a representação interna desse conhecimento que nós utilizamos criativamente para implementar a transferência de estilo! A Figura 3.29 mostra alguns exemplos gerados pelo nosso algoritmo de transferência de estilo utilizando diversas entradas e diversos estilos[9].

[9] O Apêndice A deste trabalho contém uma vasta galeria de resultados

Figura 3.29: *Linha 1*: "Praia", por Kess Streefkerk, e "A Grande Onda de Kanagawa", de Hokusai. *Linha 2*: Modelo masculino pelo fotógrafo Chris Campbell e "Os Retirantes", de Portinari. *Linha 3*: "Rádio", por Alex Blajan e "Noite Estrelada", de Van Gogh. *Linha 4*: Vêneto, na Itália, e "O Alienista", de Moon e Bá.

Esses resultados foram gerados com os seguintes pesos para os diferentes termos de perda: peso de conteúdo `c_w = 0.025`, peso de conteúdo `s_w = 5` e peso de variação total `t_v_w = 0.1`. Essa escolha veio de uma série de experimentos com diversos desses parâmetros. Os resultados da Figura 3.31 mostram um exemplo usando as imagens da Figura 3.30 e os parâmetros `c_w = 0.025`, `t_v_w = 0.1` fixos e `s_w` variando entre `0.25`, `1`, `2.5`, `5`, `7.5` e `10`.

Figura 3.30: Uma foto de Bruxelas, "a Veneza do Norte", e "Noite estrelada", de Vincent Van Gogh

Figura 3.31: Note as diferenças, especialmente na textura, conforme o peso do estilo aumenta.

Ao finalizarmos essa caminhada, você provavelmente deve ter percebido que toda essa busca por reproduzir os efeitos visuais do Prisma foi simplesmente uma arapuca para te guiar por todo esse labirinto do uso de *deep learning* para visão computacional. Mas, agora

que você chegou do outro lado, parabéns! Você não apenas aprendeu a resolver o problema da transferência de estilo e gerar belíssimos resultados, como você também adquiriu uma sólida base de conhecimento para ler, entender e reproduzir pesquisas estado da arte no campo de redes neurais convolucionais. Isso não é pouca coisa, então vá comemorar e celebrar esse grande feito.

Capítulo 4

Resultados Experimentais

Pela natureza do trabalho proposto, dois experimentos foram realizados ao longo de sua execução: a aplicação do tutorial produzido a alunos egressos da disciplina de Introdução do Processamento de Imagens, e a aplicação do algoritmo do tutorial a diversas imagens. Este capítulo detalha os resultados obtidos nesses dois experimentos.

4.1 Experimento pedagógico com uso do tutorial

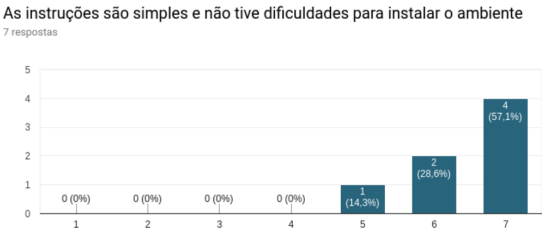
Para concluir os estudos deste trabalho, foi realizado um experimento de "aplicar" o tutorial para um grupo de sete alunos da UnB de diversos perfis. Por "aplicar" entende-se disponibilizar o conteúdo para que os próprios estudantes possam acompanhar o tutorial e executar os exercícios, evitando ao máximo a interferência por parte dos pesquisadores, para reforçar o aspecto autodidata do projeto. Quanto ao perfil dos participantes, houve um espectro que variou desde um aluno interessado em processamento de imagens, mas sem qualquer bagagem de disciplinas formais na área, até um aluno do programa de doutorado da UnB, justamente na área de *deep learning*.

4.1.1 Experimento

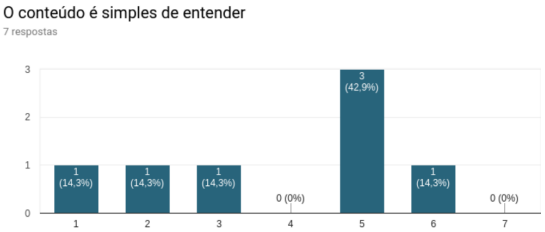
Os alunos se reuniram em um laboratório na UnB, cada um com seu computador e com acesso à internet. Eles receberam o Capítulo 3 deste trabalho em um documento à parte, para acompanharem a teoria, e receberam instruções para que pudessem montar um ambiente que os permitiria rodar os exercícios do *Jupyter Notebooks* paralelamente. Ao chegarem ao final do tutorial, foi pedido que eles respondessem um formulário *online* (detalhado no Apêndice A) que avaliaria a experiência como um todo. Em resumo, para cada aspecto do experimento que buscou-se avaliar (instruções de *setup*, conteúdo e formato do tutorial e a experiência prática em si) foram elaboradas algumas afirmações, e o aluno deveria responder de (1) a (7), variando de "Discordo completamente"a "Concordo plenamente". Ao final, havia um campo de texto livre para comentários, críticas e sugestões.

4.1.2 Resultados

Segue abaixo os gráficos das respostas do formulário.

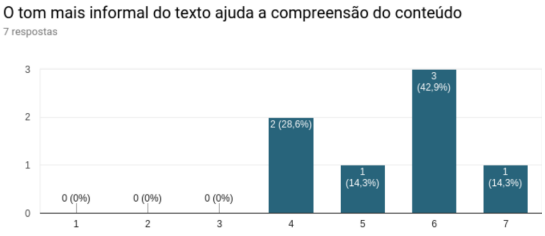


(a) Instruções sobre a instalação

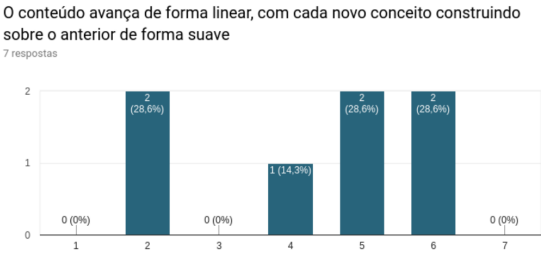


(b) Complexidade do conteúdo

Figura 4.1: Questões 1 e 2

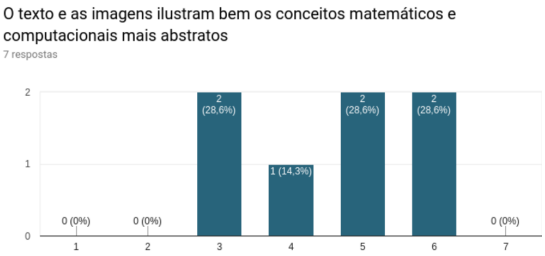


(a) Formalidade do texto

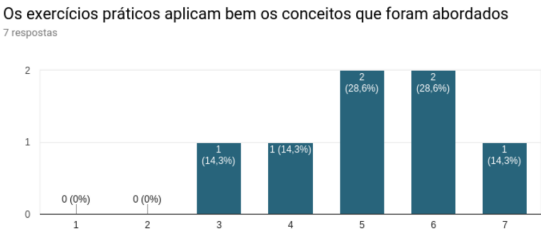


(b) Progressão do conteúdo

Figura 4.2: Questões 3 e 4

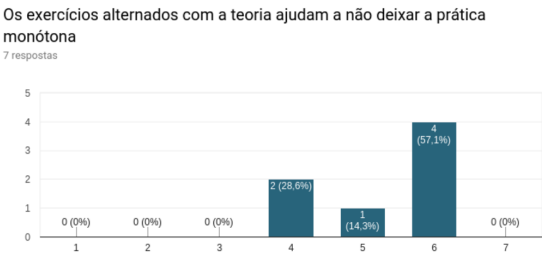


(a) Mídias auxiliares

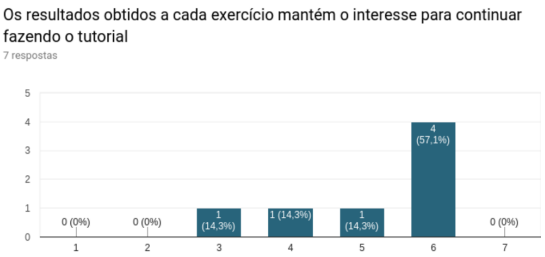


(b) Aplicação prática

Figura 4.3: Questões 5 e 6

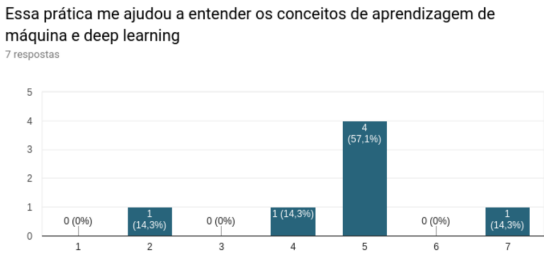


(a) Ritmo dos exercícios

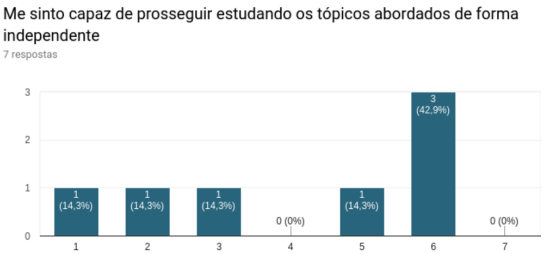


(b) Resultados dos exercícios

Figura 4.4: Questões 7 e 8

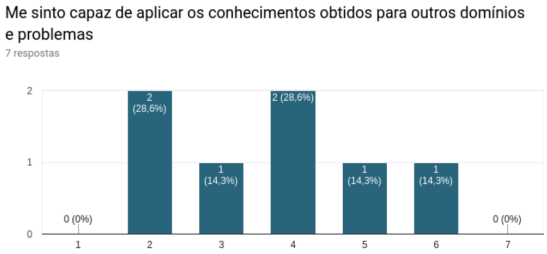


(a) Introdução aos conceitos



(b) Progressão nos estudos

Figura 4.5: Questões 9 e 10



(a) Aplicação dos conceitos

(b) Indicação do tutorial para terceiros

Figura 4.6: Questões 11 e 12

Dúvidas, críticas e comentários Resumindo o que foi respondido pelos alunos no campo de texto livre, destaca-se:

- Exercícios mais livres: permitir ao aluno trabalhar mais no código dos exercícios, ao invés de entregar o código completo, apenas para execução "passiva";
- Comentário de código: ser mais explícito na explicação de cada passo do código, uma vez que o objetivo do tutorial não é ensinar as ferramentas de programação propriamente ditas;
- Formato: centralizar em um ponto só a teoria e o exercício (incluir toda a teoria no *Jupyter*).

4.1.3 Análise e observações

Instruções de *setup* As respostas indicam que a maioria não teve problemas com a montagem do ambiente. Apenas um dos alunos, durante o experimento, não conseguia rodar os exercícios. Uma simples busca na internet pelo erro apontado pelo programa indicou a solução, realizar um *downgrade* na versão da biblioteca *TensorFlow*, o que pode ser acrescentado como observação na documentação.

Conteúdo do tutorial A primeira afirmação da pesquisa, nessa seção, foi propositalmente vaga, "O conteúdo é simples de entender", e o resultado negativo, de 50% de discordância em algum nível, se alinha com a percepção inicial do trabalho, de que os conteúdos não são simples, fazendo necessária a criação de trabalhos de facilitação didática, como este. Por outro lado, as afirmações seguintes, sobre o tom informal do texto,

o ritmo da progressão do conteúdo e o uso de recursos de imagem, e os resultados de concordância ampla quanto a elas, indicam que, pelo menos em parte, o propósito de facilitar a transmissão do conteúdo por meio das escolhas intencionais explicadas nesse trabalho foi alcançado.

Formato do tutorial A maioria de concordância nas afirmações referentes ao formato, especificamente o uso de exercícios práticos alternados com cargas de teoria, validam as hipóteses levantadas na seção de referencial teórico deste trabalho, quanto à relevância e eficácia do *Problem Based Learning*.

Experiência da atividade Nesse ponto, algumas respostas com maior discordância já apontam alguns caminhos para a evolução deste trabalho. Embora o objetivo mais básico e inicial do trabalho, de introduzir e apresentar os conceitos mais iniciais e básicos do *deep learning* tenham encontrado maioria de concordância entre as respostas, um grande número de respostas discordantes quanto à capacidade autoavaliada do aluno de prosseguir estudando e aplicando os conceitos abordados, de forma autônoma, ainda mais quando pareados com as sugestões do campo livre de texto (avaliado a seguir), indicam que algumas melhorias podem ser realizadas para alcançar de forma mais eficaz o objetivo de empoderamento do aluno, proposto neste trabalho. Apesar disso, para satisfação pessoal dos proponentes do trabalho, houve ampla concordância (mais da metade das respostas (7)) quanto ao potencial do trabalho, indicada nas respostas à afirmação sobre indicar o trabalho para outros alunos, afirmando e confirmando positivamente a hipótese central deste trabalho, de que tutoriais dinâmicos, esteticamente atraentes e autocontidos podem auxiliar a tarefa docente em relação a tópicos mais avançados das disciplinas de processamento digital de imagens e visão computacional e enriquecer a experiência de aprendizagem por parte do aluno.

Melhorias futuras Devido ao espectro amplo do perfil dos alunos que participaram do experimento, já detalhado antes, vários comentários foram em direções opostas: o conteúdo pode ser mais superficial, ou mais profundo; o código dos exercícios poderia ser mais simples, mais abstraído, ou mais profundo e mais detalhado. Quanto a esses casos, as decisões tomadas, principalmente em relação ao nível de profundidade do conteúdo, foram baseadas no perfil do aluno egresso da disciplina de Introdução ao Processamento de Imagens junto ao professor da mesma, mas podem ser revistas caso esse perfil se altere com o tempo, ou o público-alvo da prática do tutorial seja alterado.

Para o aprimoramento e evolução deste trabalho, assim como a eventual elaboração de outro trabalhos semelhantes que abordem outros tópicos ou áreas, algumas sugestões certamente podem levar a resultados ainda mais positivos. Dentre essas, se destacam a forma da experiência, de disponibilizar toda a teoria do tutorial no mesmo ambiente dos experimentos, no *Jupyter*, e tornar as partes práticas ainda mais interativas, incentivando o aluno a alterar e experimentar o código disponível (uma vez que isso já é possível, mas muitos nem sequer cogitaram essa possibilidade), ou mesmo codificar completamente algumas seções, tendo mais controle sobre o código que será executado.

4.2 Experimento de Transferência de Estilo

Durante o desenvolvimento do trabalho, foram gerados diversos resultados do algoritmo apresentado[1]. Para essa seleção, foi realizada uma pesquisa *online* (detalhada no Apêndice A), e eles são apresentados aqui.

4.2.1 Experimento

Para gerar resultados diversos, que pudessem capturar e representar as possibilidades amplas do algoritmo, foram escolhidas 5 fotos dentro de três categorias: *paisagens*, *objetos* e *retratos*[2], conforme a Figura 4.7. Essas fotos foram usadas como *conteúdo* do experimento.

Figura 4.7: Divididos por linha: Paisagem P1, P2, P3, P4 e P5, Objeto O1, O2, O3, O4 e O5, e Retrato R1, R2, R3, R4 e R5

Para o *estilo*, foram escolhidas 9 obras de autores distintos, separados em dois grupos: *artistas brasileiros* e *artistas internacionais*, de épocas, estilos e escolas variados, conforme a Figura 4.8.

Cada imagem de conteúdo foi submetida ao algoritmo pareada com cada um dos estilos, gerando um total de 135 resultados. Após uma pesquisa *online* (detalhada no Apêndice A), foram selecionados alguns destaques apresentados aqui.

Figura 4.8: *Linha 1*: Artistas brasileiros, Romero Britto, Beatriz Milhazes, os gêmeos Fábio Moon e Gabriel Bá, e Cândido Portinari *Linha 2*: Artistas internacionais: Caravaggio, Gustav Doré, Pablo Picasso, Vincent Van Gogh e mestre Hokusai

4.2.2 Resultados

Observações

Com base nas notas médias obtidas por cada imagem, seguem alguns dados tabulados:

Imagem	**Nota média**
Moon e Bá - Paisagem P5	4,89
Hokusai - Paisagem P1	4,70
Van Gogh - Paisagem P1	4,65
Hokusai - Paisagem P4	4,50
Hokusai - Retrato R3	4,43
Moon e Bá - Paisagem P1	4,42
Moon e Bá - Objeto O1	4,36
Portinari - Objeto O3	4,36
Portinari - Paisagem P5	4,33
Van Gogh - Paisagem P4	4,33

Tabela 4.1: 10 melhores resultados gerais

[1]Um agradecimento especial ao projeto Intel®AI DevCloud, que permitiu a execução desses experimentos em tempo hábil. Mais em: https://software.intel.com/pt-br/ai-academy/tools/devcloud

[2]Retiradas de um banco de imagens gratuito, https://unsplash.com/

Figura 4.9: Categoria **Paisagem**, melhores e piores. *Linha 1*: Médias de avaliação: 4,89, 4,70 e 4,65 *Linha 2*: Médias de avaliação: 2,56, 2,87 e 2,91

Figura 4.10: Categoria **Objeto**, melhores e piores. *Linha 1*: Médias de avaliação: 4,36, 4,36 e 4,25 *Linha 2*: Médias de avaliação: 3,26, 3,28 e 3,48

Figura 4.11: Categoria **Retrato**, melhores e piores. *Linha 1*: Médias de avaliação: 4,43, 4,25 e 4,23 *Linha 2*: Médias de avaliação: 3,08, 3,16 e 3,28

Imagem	**Nota média**
Milhazes - Paisagem P4	2,56
Milhazes - Paisagem P3	2,87
Britto - Paisagem P3	2,91
Caravaggio - Paisagem P4	3,03
Caravaggio - Retrato R2	3,08
Britto - Paisagem P1	3,09
Portinari - Retrato R5	3,16
Milhazes - Paisagem P1	3,20
Britto - Paisagem P2	3,22
Caravaggio - Paisagem P3	3,23

Tabela 4.2: 10 piores resultados gerais

Artista	**Nota média**
Hokusai	4,21
Moon e Bá	4,07
Van Gogh	3,91
Picasso	3,85
Portinari	3,82
Doré	3,81
Caravaggio	3,44
Britto	3,42
Milhazes	3,38

Tabela 4.3: Média geral de cada artista

Capítulo 5

Conclusão

Dadas as questões iniciais levantadas na introdução, da importância e relevância das técnicas de *deep learning*, e a dificuldade para tratar desse assunto nas disciplinas de Introdução ao Processamento de imagens, buscou-se elaborar um tutorial de aplicação de uma aplicação específica de redes neurais para introduzir e expandir o assunto de uma forma que pudesse ser estudada pelo aluno de forma independente. "Nas artes visuais, especialmente na pintura, os seres humanos dominaram a habilidade de criar experiências visuais, por meio de compor uma interação complexa entre o conteúdo e o estilo de uma imagem", afirmam Gatys *et al.* no artigo que inspirou e fundamentou as técnicas usadas e explicadas no artefato produzido como resultado desse trabalho. O que o algoritmo aqui apresentado foi capaz de mostrar é que as técnicas de visão computacional estado da arte, em especial das redes neurais convolucionais, começam se mostrar opções viáveis para reproduzir de forma automatizada alguns dos aspectos mais subjetivos do processo criativo artístico como descrito por Gatys *et al.*

Tendo em vista o objetivo educacional desse trabalho, o tutorial desenvolvido se baseia na efetividade da forma, alternando texto didático e atividades práticas de retorno imediato, e do conteúdo, usando o apelo visual e estético de grandes obras de arte para inspirar e motivar os alunos que eventualmente utilizarão este artefato. O experimento realizado com alguns alunos, detalhado no Capítulo 4, obteve resultados, no geral, positivos, indicando que os objetivos mais básicos deste trabalho, de produzir um material didático que amplie e enriqueça a experiência de aprendizado ao introduzir conceitos de *deep learning* de forma autônoma, por parte do aluno, foram alcançados, e a propensão da maioria de indicar a prática do tutorial para terceiros indica que os próprios alunos reconhecem o valor, a importância e o potencial didático da experiência. Por outro lado, ainda que não de forma negativa, alguns pontos de melhoria foram levantados pelos alunos e alertam para a necessidade de serem refinados ou retrabalhados, em especial à forma de apresentação da prática e a interação do aluno com os exercícios propostos. Um aprofundamento nas boas práticas no campo do *Problem Based Learning* certamente podem ser úteis para a melhoria desses aspectos.

Como já foi demonstrado na seção de resultados, o resultado do algoritmo usado no tutorial como motivado visual e estético é bastante eficaz. De todas as imagens que foram incluídas na pesquisa, apenas 3 tiveram nota abaixo da média 2,5, e grande parte obteve média acima de 4, mostrando como os resultados são considerados esteticamente agradáveis. Nesse sentido, a escolha de uma técnica de alto apelo estético se mostra

acertada, pois motiva o aluno a querer participar do tutorial, como também serve de recompensa para o trabalho empregado no estudo e até mesmo como fator de publicidade do experimento("olha que legal o que eu fiz!").

Sobre possibilidades de melhoria e evolução do presente trabalho, um grande problema do algoritmo apresentado é ser bastante demorado, em um computador comum, para gerar os resultados finais. Enquanto o aplicativo *Prisma*, que citamos no início do tutorial, leva alguns segundos para gerar as imagens estilizadas, o algoritmo apresentado pode levar dezenas (de centenas) de segundos por iteração em um computador sem uma boa GPU dedicada. Para resolver esse problema de velocidade, uma saída seria implementar a ideia brilhante introduzida em Johnson *et al* [37]. Nesse artigo, ao invés de resolver o caríssimo (em termos computacionais) problema de otimização de transferência de estilo, eles treinam uma outra *convnet*, à parte, para otimizar os resultados da solução. Uma vez treinado um modelo para uma imagem de estilo específica, ainda assim um processo custoso, ele pode ser aplicado em questões de segundos a qualquer imagem de conteúdo, aumentado em até 1000 vezes a velocidade do processo.

Por fim, esperamos que o artefato produzido nesse trabalho seja útil no ensino das técnicas e teorias nele apresentadas e estimule e facilite a propagação das técnicas de *deep learning* no meio acadêmico em nível de graduação.

Para detalhes da metodologia dos experimentos, ver o Apêndice A. No Apêndice B foi incluído um experimento à parte, utilizando o famoso teste de Rorschach como conteúdo e imagens de Salvador Dalí como estilo.

Referências

[1] Wilhelm Burger e Mark J. Burge. *Digital Image Processing – An Algorithmic Introduction using Java*. Springer-Verlag, 2011. 1

[2] Salman Khan e Hossein Rahmani e Syed Afaq Ali Shah e Mohammed Bennamoun. *A Guide to Convolutional Neural Networks for Computer Vision*. Morgan & Claypool Publishers, 2018. 1, 5

[3] Dipanjan Sarkar e Raghav Bali e Tushar Sharma. *Practical Machine Learning with Python*. Apress, 2018. 1, 8

[4] Sociedade Brasileira de Computação. Currículo de referência - cc e ec versão:2005. http://www.sbc.org.br/index.php?option=com_jdownloads&Itemid=195&task=view.download&catid=36&cid=183, 2005. [Acesso em 15 Maio 2018]. 1

[5] Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira. Provas e gabaritos: 2005. http://portal.inep.gov.br/web/guest/enade/provas-e-gabaritos-2005, 2005. [Acesso em 15 Maio 2018]. 1

[6] Ministério da Educação Conselho Nacional de Educação. Referenciais de formação para cursos de graduacao em computação. http://www.sbc.org.br/documentos-da-sbc/send/131-curriculos-de-referencia/1165-referenciais-de-formacao-para-cursos-de-graduacao-em-computacao-outubro-2017 2017. [Acesso em 15 Maio 2018]. 1, 2, 6

[7] Maria Petrou e Costas Petrou. *Image Processing: The Fundamentals*. Wiley, 2010. 3

[8] Rafael C. Gonzales e Richard E. Woods. *Digital Image Processing*. Prentice Hall, 2001. 3

[9] Al Bovik. *The Essential Guide to Image Processing*. Academic Press, 2009. 4

[10] L2 Intelligence Reports. Instagram 2014. http://www.l2thinktank.com/research/instagram-2014, 2014. [Acesso em 3 Setembro 2014]. 4

[11] François Chollet. *Deep learning with Python*. Manning Publications Co., 2018. 4, 5

[12] Alan Turing. Computing machinery and intelligence. *Mind*, LIX(236):433–460, 1950. 4

[13] Donald R. Woods. Problem-based learning for large classes in chemical engineering. *New Directions for Teaching and Learning*, 1996(68):91–99, 1996. 6

[14] Thomas Kluyver e Benjamin Ragan-Kelley e Fernando Pérez e Brian Granger e Matthias Bussonnier e Jonathan Frederic e Kyle Kelley e Jessica Hamrick e Jason Grout e Sylvain Corlay e Paul Ivanov e Damián Avila e Safia Abdalla e Carol Willing e Jupyter development team. Jupyter notebooks - a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016. 7

[15] Kenneth E. Iverson. *A Programming Language*. John Wiley & Sons, Inc., 1962. 7

[16] Connie Malamed. *Visual Design Solutions*. John Wiley & Sons, 2015. 8

[17] Alice M. Isen. Missing in action in the aim: Positive affect's facilitation of cognitive flexibility, innovation, and problem solving. *Psychological Inquiry*, 13(1):57–65, 2002. 8

[18] Jan Plass e Steffi Heidig e Elizabeth Hayward e Bruce Homer e Enjoon Um. Emotional design in multimedia learning: Effects of shape and color on affect and learning. 29:128–140, 2014. 8

[19] Steffi Heidig e Julia Müller e Maria Reichelt. Emotional design in multimedia learning. *Comput. Hum. Behav.*, 44(C):81–95, 2015. 9

[20] Enjoon Um e Jan Plass e Elizabeth Hayward e Bruce Homer. Emotional design in multimedia learning. 104:485–498, 2012. 9

[21] Donald A. Norman. *Emotional design*. Basic Books, 2004. 9

[22] Fei-Fei Li e Andrej Karpathy e Justin Johnson. Cs231n: Convolutional neural networks for visual recognition 2016. 2016. 10, 16, 28, 29, 30

[23] Leon A. Gatys e Alexander S. Ecker e Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015. 10, 13, 14

[24] François Chollet e outros. Keras. https://keras.io, 2015. 10, 38

[25] Jackie Dove. Prisma review: Intelligent photo effects app taps into deep learning for an edgy art connection. https://www.macworld.com/article/3107269/software-graphics/prisma-review-intelligent-photo-filter-app-taps-into-deep-learning-for-an-edgy-ar.html, 2016. [Acesso em 15 Maio 2018]. 12

[26] Alphonse Mucha. Gismonda. https://en.wikipedia.org/wiki/List_of_works_by_Alphonse_Mucha#/media/File:Alfons_Mucha_-_1894_-_Gismonda.jpg, 1894. [Acesso em 15 Maio 2018]. 12

[27] Nigel Tadyanehondo. Tate modern, london, united kingdom. https://unsplash.com/photos/ic0N8P5dtCY, 2017. [Acesso em 15 Maio 2018]. 12

[28] Ian Goodfellow e Yoshua Bengio e Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 18, 33

[29] Andreas Griewank. *Evaluating derivatives*. Society for Industrial and Applied Mathematics, 2000. 18

[30] A guide to tf layers: Building a convolutional neural network. https://www.tensorflow.org/tutorials/layers, 2018. [Acesso em 15 Maio 2018]. 19, 23

[31] Yann LeCun e Corinna Cortes. MNIST handwritten digit database. 2010. 19

[32] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. 23

[33] Perceptrons - the most basic form of a neural network. https://appliedgo.net/perceptron/, 2016. [Acesso em 15 Maio 2018]. 24

[34] Karen Simonyan e Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 34, 35, 40

[35] Jia Deng e Wei Dong e Richard Socher e Li-jia Li e Kai Li e Li Fei-fei. Imagenet: A large-scale hierarchical image database. In *In CVPR*, 2009. 35

[36] Justin Johnson. neural-style. https://github.com/jcjohnson/neural-style, 2015. 38

[37] Justin Johnson e Alexandre Alahi e Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016. 40, 60

[38] Guillaume Berger e Roland Memisevic. Incorporating long-range consistency in cnn-based texture generation. *CoRR*, abs/1606.01286, 2016. 43

[39] Susana Martinez-Conde e Dave Conley e Hank Hine e Joan Kropf e Peter Tush e Andrea Ayala e Stephen L. Macknik. Marvels of illusion: illusion and perception in the art of salvador dali. *Frontiers in Human Neuroscience*, 9, 2015. 72

[40] João Maria do Amaral Torres. O teste rorschach na história da avaliação psicológica. *Revista do NUFEN*, 2:92 – 104, 06 2010. 72

Apêndice A

Metodologia de pesquisa

Para medir os resultados do trabalho, foram elaboradas duas pequisas, uma para avaliar a aplicação do tutorial em ambiente acadêmico, e outra para avaliar a eficácia do algoritmo usado no tutorial. Em ambos, a ferramenta utilizada para a medição foi o *Google Forms*, ferramenta de criação *online* de formulários que permite criar pesquisas rapidamente, e salva os resultados em planilhas eletrônicas, facilitando a análise posterior dos dados coletados. Ambos os formulários foram aplicados de forma anônima.

A.1 Questionário sobre aplicação do tutorial de Transferência de Estilo Artístico

Todos os passos do questionário foram reproduzidos nas Figuras A.1, A.2, A.3, A.4 e A.6

A.2 Questionário de Avaliação de Transferência de Estilo

Por se tratar de um questionário mais longo, esse formulário foi dividido em duas partes, uma para os artistas brasileiros e outro para os artistas internacionais. Para cada resultado gerado pelo algoritmo, foi exibido o conteúdo, o estilo e o resultado, e o usuário deveria responder, em uma escala de 1 a 5, o quanto o resulto o agradava.

Por se tratar de um questionário bem mais longo que o anterior, ainda mais que foram inseridos largos espaçamentos entre as imagens, para que o usuário pudesse avaliar uma imagem de cada vez, sem interferência das outras, aqui está reproduzido apenas uma parte de um desses formulários. Todo o restante segue esse mesmo modelo, nas duas partes dele.

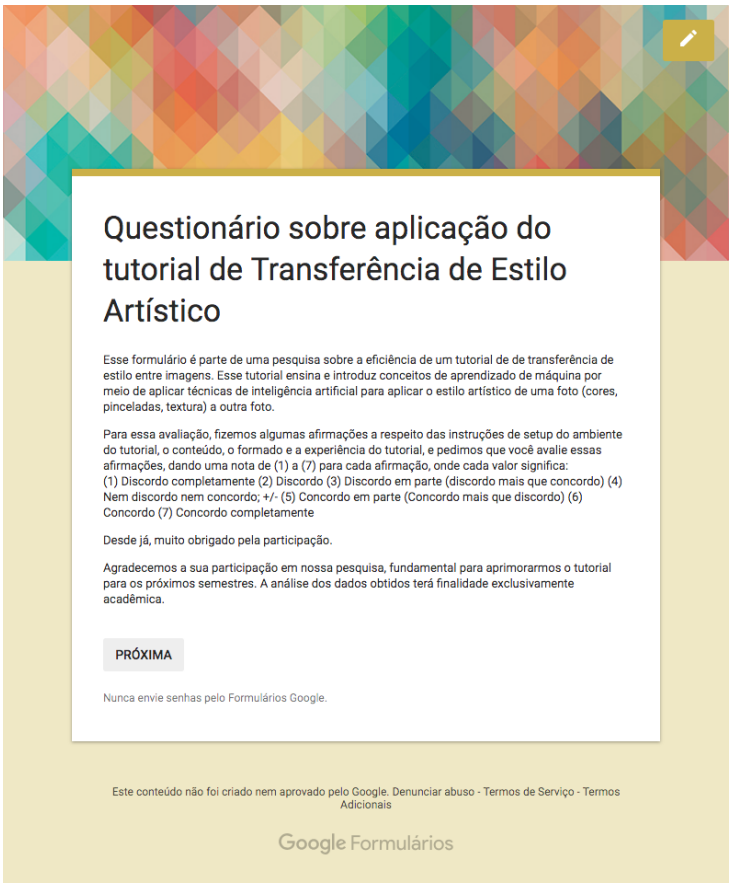


Figura A.1: Tela de apresentação do formulário de pesquisa

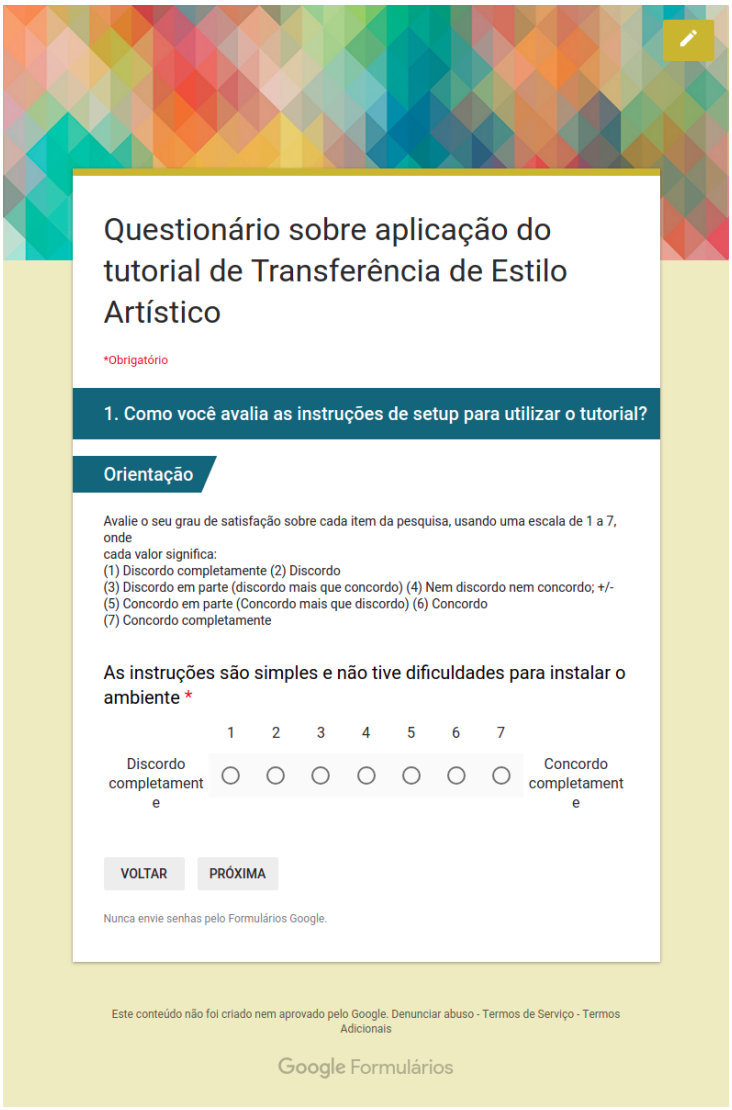


Figura A.2: Avaliação das instruções iniciais de montagem de ambiente do tutorial

Questionário sobre aplicação do tutorial de Transferência de Estilo Artístico

*Obrigatório

2. Como você avalia o conteúdo do tutorial?

Orientação

Avalie o seu grau de satisfação sobre cada item da pesquisa, usando uma escala de 1 a 7, onde
cada valor significa:
(1) Discordo completamente (2) Discordo
(3) Discordo em parte (discordo mais que concordo) (4) Nem discordo nem concordo; +/-
(5) Concordo em parte (Concordo mais que discordo) (6) Concordo
(7) Concordo completamente

O conteúdo é simples de entender *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

O tom mais informal do texto ajuda a compreensão do conteúdo *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

O conteúdo avança de forma linear, com cada novo conceito construindo sobre o anterior de forma suave *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

O texto e as imagens ilustram bem os conceitos matemáticos e computacionais mais abstratos *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

VOLTAR PRÓXIMA

Nunca envie senhas pelo Formulários Google.

Este conteúdo não foi criado nem aprovado pelo Google. Denunciar abuso - Termos de Serviço - Termos Adicionais

Google Formulários

Questionário sobre aplicação do tutorial de Transferência de Estilo Artístico

*Obrigatório

3. Como você avalia o formato do tutorial?

Orientação

Avalie o seu grau de satisfação sobre cada item da pesquisa, usando uma escala de 1 a 7, onde
cada valor significa:
(1) Discordo completamente (2) Discordo
(3) Discordo em parte (discordo mais que concordo) (4) Nem discordo nem concordo; +/-
(5) Concordo em parte (Concordo mais que discordo) (6) Concordo
(7) Concordo completamente

Os exercícios práticos aplicam bem os conceitos que foram abordados *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

Os exercícios alternados com a teoria ajudam a não deixar a prática monótona *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

Os resultados obtidos a cada exercício mantém o interesse para continuar fazendo o tutorial *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

VOLTAR PRÓXIMA

Nunca envie senhas pelo Formulários Google.

Este conteúdo não foi criado nem aprovado pelo Google. Denunciar abuso - Termos de Serviço - Termos Adicionais

Google Formulários

Questionário sobre aplicação do tutorial de Transferência de Estilo Artístico

*Obrigatório

4. Como você avalia a sua participação na atividade com o tutorial?

Orientação

Avalie o seu grau de satisfação sobre cada item da pesquisa, usando uma escala de 1 a 7, onde
cada valor significa:
(1) Discordo completamente (2) Discordo
(3) Discordo em parte (discordo mais que concordo) (4) Nem discordo nem concordo; +/-
(5) Concordo em parte (Concordo mais que discordo) (6) Concordo
(7) Concordo completamente

Essa prática me ajudou a entender os conceitos de aprendizagem de máquina e deep learning *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

Me sinto capaz de prosseguir estudando os tópicos abordados de forma independente *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

Me sinto capaz de aplicar os conhecimentos obtidos para outros domínios e problemas *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

Pretendo indicar essa prática para outros alunos interessados em processamento de imagens e visão computacional *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

VOLTAR PRÓXIMA

Nunca envie senhas pelo Formulários Google.

Questionário sobre aplicação do tutorial de Transferência de Estilo Artístico

*Obrigatório

4. Como você avalia a sua participação na atividade com o tutorial?

Orientação

Avalie o seu grau de satisfação sobre cada item da pesquisa, usando uma escala de 1 a 7, onde
cada valor significa:
(1) Discordo completamente (2) Discordo
(3) Discordo em parte (discordo mais que concordo) (4) Nem discordo nem concordo; +/-
(5) Concordo em parte (Concordo mais que discordo) (6) Concordo
(7) Concordo completamente

Essa prática me ajudou a entender os conceitos de aprendizagem de máquina e deep learning *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

Me sinto capaz de prosseguir estudando os tópicos abordados de forma independente *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

Me sinto capaz de aplicar os conhecimentos obtidos para outros domínios e problemas *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

Pretendo indicar essa prática para outros alunos interessados em processamento de imagens e visão computacional *

	1	2	3	4	5	6	7	
Discordo completamente	○	○	○	○	○	○	○	Concordo completamente

VOLTAR PRÓXIMA

Nunca envie senhas pelo Formulários Google.

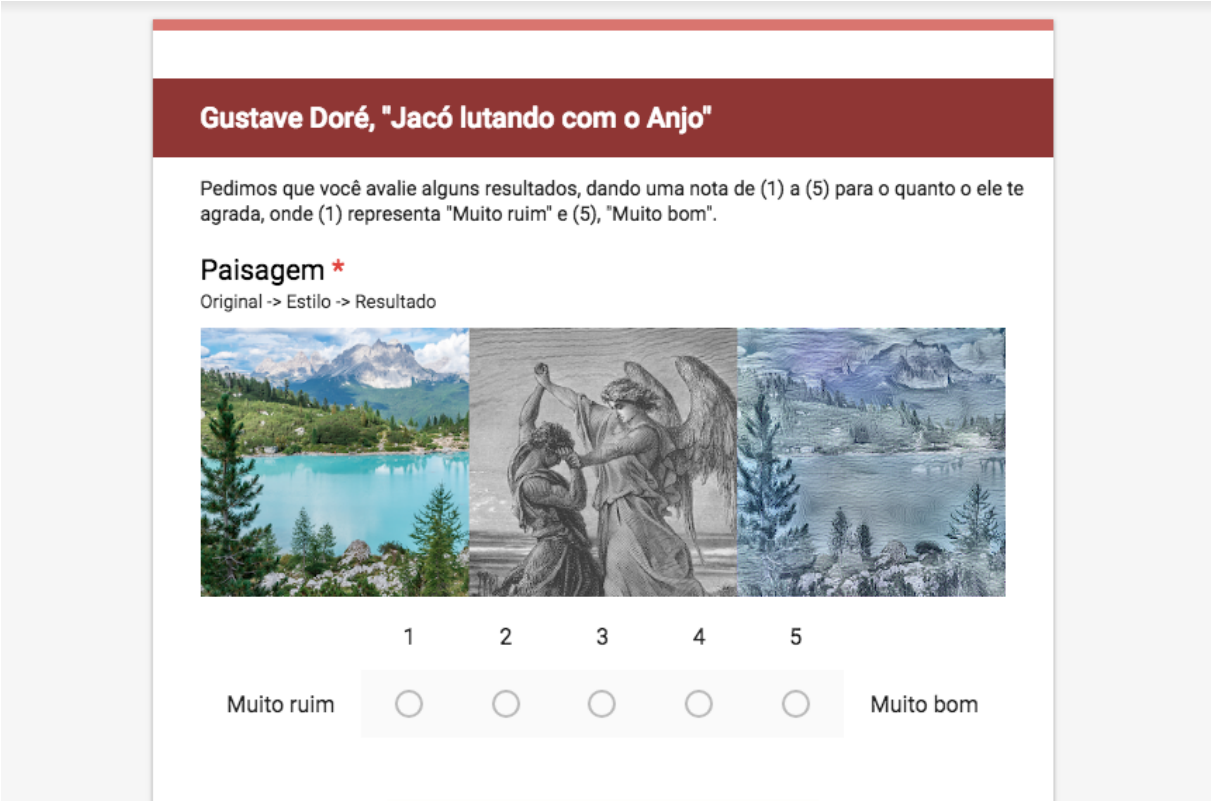


Figura A.7: Exemplo de questão do formulário

Apêndice B

Rorschach e Dalí

Durante a fase de seleção dos artistas que seriam usados como estilos para esse trabalho, o nome de Salvador Dalí foi levantado diversas vezes. Após alguns resultados, observou-se que, embora os temas de suas obras sejam notoriamente famosos, suas técnicas não geram resultados de transferência de estilo tão chamativos em fotos comuns, uma vez que as características perceptivas mais focadas pelo algoritmo apresentado nesse trabalho, como técnica de pintura e texturas, não eram o foco de Dalí.

Salvador Dalí entendia que o que nós construímos visualmente como realidade é o produto de hábitos da mente, mais do que do olho. Ele entendia que o ser humano cria um mundo ordenado ou desordenado a partir de informações intermitentes e incompletas da retina, processadas pelas experiências, desejos e apreensões da mente. Assim, as obras de Dali desafiam as percepções do observador a respeito da realidade e permitem que ele veja além da superfície [39].

A partir disso, desse forte aspecto psicológico das obras do pintor catalão, surgiu a ideia de, como um complemento ao trabalho, utilizar o estilo de Dalí em um tipo diferente de imagens de conteúdo. Se Dalí se destacou como o pintor das experiências da mente, nenhum experimento psicológico talvez seja tão popularmente conhecido como o *Teste de Rorschach*.

O teste de Rorschach é um método de avaliação psicológica desenvolvido pelo psiquiatra suíço Hermann Rorschach, publicado em 1921, em que "o indivíduo é induzido a revelar seu mundo privado, expressando o que vê em diversas lâminas, onde pode projetar seus sentimentos, justamente porque as lâminas não constituem objetos socialmente estandardizados, ou situações frente as que deve dar respostas culturalmente aceitas" [40].

Se o estilo de Dalí combina tanto, na teoria, com o teste de Rorschach, a avaliação da combinação visual dos dois fica a cargo do leitor, que pode observar esse experimento a seguir. Nos resultados aqui apresentados, *o que você vê?*

B.1 Experimento

Para o algoritmo desse trabalho, foram escolhidas as dez pranchas do Teste de Rorschach, apresentadas na Figura B.1, para serem o *conteúdo*.

Das obras de Dalí, foram selecionados duas pinturas para serem o estilo, "Criança geopolítica observando o nascimento do homem novo" e "Galatea das Esferas", apresentadas na Figura B.2.

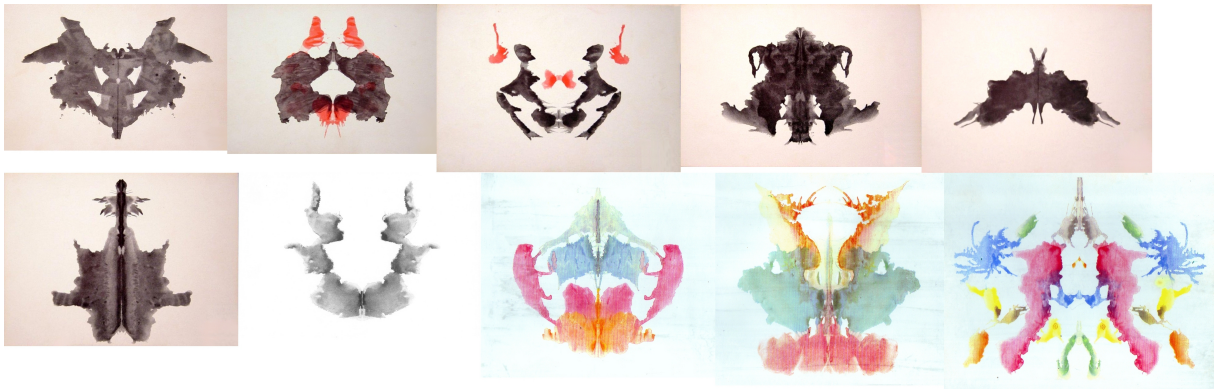


Figura B.1: As dez pranchas do Teste de Rorschach

Figura B.2: "Criança geopolítica observando o nascimento do homem novo" e "Galatea das Esferas", de Salvador Dalí

B.2 Resultados

Figura B.3: Pranchas de Rorschach (1-5), "Criança geopolítica observando o nascimento do homem novo" e "Galatea das Esferas"

Figura B.4: Pranchas de Rorschach (6-10), "Criança geopolítica observando o nascimento do homem novo" e "Galatea das Esferas"